공학석사 학위논문

LSTM Language Modeling for Intrusion Detection Systems

LSTM 기반 언어 모델을 통한 침입 탐지 시스템

2017년 8월

서울대학교 대학원

전기·정보공학부

김규완

LSTM Language Modeling for Intrusion Detection Systems

지도교수 윤 성 로

이 논문을 공학석사 학위논문으로 제출함

2017년 8월

서울대학교 대학원

전기·정보공학부

김규완

김규완의 공학석사 학위논문을 인준함

2017년 8월

위 원 장 백 윤 흥 (인)

부위원장 윤 성 로 (인)

위 원 하 정 우 (인)

국문초록

컴퓨터 보안에서 견고한 침입 탐지 시스템을 설계하는 것은 가장 핵심적이고 중요한 문제 중의 하나이다. 본 논문에서는 비정상 기반 호스트 침입 탐지 시스템 설계를 위한 시스템 콜 시퀀스와 분기 시퀀스에 대한 언어 모델 방법을 제안한다. 기존의 방법에서 흔히 발생하는 높은 오탐율 문제를 해결하기 위해 여러 임계값 분류기를 혼합하여 정상적인 시퀀스들을 잘 모을 수 있는 새로운 앙상블 방법을 사용하였다. 본 언어 모델은 기존 방법들이 잘 하지 못했던 각 시스템 콜의 의미와 그들 간의 상호 작용을 학습 할 수 있다는 장점이 있다. 공개된 데이터들과 새롭게 생성한 데이터를 바탕으로 다양한 실험을 통해 제안 된 방법의 타당성과 유효성을 입증하였다. 또한, 본 모델이 높은 이식성을 갖고 있음을 보였다.

주요어 : 컴퓨터 보안, 언어 모델, 이상 탐지, LSTM, 딥러닝, 앙상블

학 번 : 2015-20891

Contents

List of Tables

List of Figures

Chapter 1 Introduction

An intrusion detection system (IDS) (a.k.a program anomaly detection) refers to a hardware/software platform for monitoring network or system activities to detect malicious signs therefrom. Nowadays, practically all existing computer systems operate in a networked environment, which continuously makes them vulnerable to a variety of malicious activities. Over the years, the number of intrusion events is significantly increasing across the world, and intrusion detection systems have already become one of the most critical components in computer security. With the explosive growth of logging data, the role of machine learning in effective discrimination between malicious and benign program activities has never been more important.

A survey of existing IDS approaches needs a multidimensional consideration. Depending on the scope of intrusion monitoring, there exist two main types of intrusion detection systems: network-based (NIDS) and host-based (HIDS). The network-based intrusion detection systems monitor communications between hosts, while the host-based intrusion detection systems monitor the activity on a single system. From a methodological point of view, intrusion detection systems can also be classified into two classes [1]: signature-based and anomaly-based. The signature-based approaches match the observed behaviors against templates of known attack patterns, while the anomaly-based techniques compare the observed behaviors against an extensive baseline of normal behaviors constructed from prior knowledge, declaring each of anomalous activities to be an attack. The signature-based methods detect already known and learned attack patterns well but have an innate difficulty in detecting unfamiliar attack patterns. On the other hand, the anomaly-based methods can potentially detect previously unseen attacks but may

suffer from making a robust baseline of normal behavior, often yielding high false alarm rates. Sometimes, the distinction between what is normal or not is unclear. The ability to detect a 'zero-day' attack (i.e., vulnerability unknown to system developers) in a robust manner is becoming an important requirement of an anomaly-based approach. In terms of this two-dimensional taxonomy, we can classify our proposed method as an anomaly-based host intrusion detection system.

Building a robust intrusion detection system against any possible attacks is a very challenging task. Historically, attackers and defenders were both developed to get over each other. If attacks using loopholes in an existing system is addressed, new defence mechanisms to prevent them need to be invented, and so on. Recently, as deep learning approches brought great improvement in many applications, a lot of security papers using deep learning are also published and my work is one of them.

It was Forrest et al. [2] who first started to use system-call traces as the raw data for host-based anomaly intrusion detection systems, and system-call traces have been widely used for IDS research and development since their seminal work [3]. Recently, [4] proposed to use neural networks on top of a sequence of system calls in the context of HIDS. System calls represent low-level interactions between programs and the kernel in the system, and many researchers consider system-call traces as the most accurate source useful for detecting intrusion in an anomaly-based HIDS. From a data acquisition point of view, system-call traces are easy to collect in a large quantity in real-time. Our approach also utilizes system-call traces as input data.

For nearly two decades, various research has been conducted based on analyzing system-call traces. Most of the existing anomaly-based host intrusion detection methods typically aim to identify meaningful features using the frequency of individual calls and/or windowed patterns of calls from sequences of system calls. However, such methods have limited ability to capture call-level features and phrase-

level features simultaneously. As will be detailed shortly, our approach tries to address this limitation by generating a language model of system calls that can jointly learn the semantics of individual system calls and their interactions (that can collectively represent a new meaning) appearing in call sequences.

In natural language processing (NLP), a language model represents a probability distribution over sequences of words, and language modeling has been a very important component of many NLP applications, including machine translation [5, 6], speech recognition [7], question answering [8], and summarization [9]. Recently, deep recurrent neural network (RNN)-based language models are showing remarkable performance in various tasks [10, 11]. It is expected that such neural language models will be applicable to not only NLP applications but also signal processing, bioinformatics, economic forecasting, and other tasks that require effective temporal modeling.

Motivated by this performance advantage and versatility of deep RNN-based language modeling, we propose an application of neural language modeling to host-based intrusion detection. We consider system-call sequences as a language used for communication between users (or programs) and the system. In this view, system calls and system-call sequences correspond to words and sentences in natural languages, respectively. Based on this system-call language model, we can perform various tasks that comprise our algorithm to detect anomalous system-call sequences: e.g., estimation of the relative likelihood of different words (i.e., system calls) and phrases (i.e., a window of system calls) in different contexts.

The idea of using artificial neural networks for IDSs has been popular [4, 12-15]. For more recent deep learning-based techniques, there exists an example that utilized LSTM for improving intrusion detection performance [16, 17]. However, the work by [16, 17] was in essence a feature-based supervised classifier (rather than

an anomaly detector) requiring heavy annotation efforts to create labels. As such, their work required explicitly labeled attack data and possessed an inherent limitation that it could not detect new types of attacks. In addition, their approach was not an end-to-end framework and needed careful feature engineering to extract salient features for the classification task. Only one binary label was given per sequence to train their model, unlike our proposed method that is trained to predict the next call, effectively capturing contextual information needed for classification.

As many attacks are invented to detour system call based intrusion detection systems, a need to utilize additional information is raised [4, 18-20]. We make an extension from a system call language model to a branch language model. We collected logs of branches with low runtime overhead while programs are executed and used them for training and evaluation of the branch language model.

Specific contributions of this thesis can be summarized as follows: First, to model sequences of system calls (or branches), we propose a neural language modeling technique that utilizes long short-term memory (LSTM) [21] units for enhanced long-range dependence learning. The present work is one of the first end-to-end frameworks to model system-call sequences as a natural language for effectively detecting anomalous patterns therefrom. Second, to reduce false-alarm rates of anomaly-based intrusion detection, we propose a leaky rectified linear units (ReLU) [22] based ensemble method that constructs an integrative classifier using multiple (relatively weak) thresholding classifiers. Each of the component classifiers is trained to detect different types of 'highly normal' sequences (i.e., system call sequences with very high probability of being normal), and our ensemble method blends them to produce a robust classifier that delivers significantly lower false-alarm rates than other commonly used ensemble methods. These two aspects of our contributions can seamlessly be combined into a single framework. Note that the

ensemble method we propose is not limited to our language-model based front-end but also applicable to other types of front-ends. Finally, we expect our generalization to branch language model could detect more sophisticated contemporary attacks.

In the rest of this thesis, we will explain more details of our approach and then present our experimental results that demonstrate the effectiveness of our proposed method. Also, we will describe the direction of future research and make a conclusion on this thesis.

Chapter 2 Language Model of System Call Sequences

2.1 Model Architecture

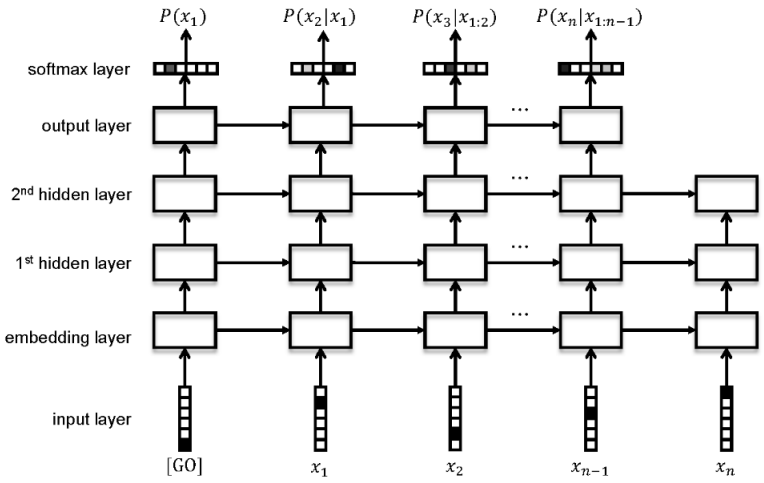

Figure 1: LSTM language model architecture.

Figure 2 illustrates the architecture of LSTM language model. The system call language model estimates the probability distribution of the next call in a sequence given the sequence of previous calls. Let vocabulary V be the set of all possible branch target addresses. The host system generates a finite number of system calls. Then, each system call is indexed by an integer starting from 1 to K(= |V|). Let $x = x_1 x_2 \cdots x_l (x_i \in V)$ denote a sequence of l system calls.

At the input layer, the call at each time step x_i is fed into the model in the form of one-hot encoding, in other words, a K dimensional vector with all elements zero except position x_i. At the embedding layer, incoming calls are embedded to continuous space by multiplying embedding matrix W, which should be learned. At the hidden layer, the LSTM unit has an internal state, and this state is updated recurrently at each time step. At the output layer, a softmax activation function is

used to produce the estimation of normalized probability values of possible calls coming next in the sequence, $P(x_i|x_{1:i-1})$. According to the chain rule, we can estimate the sequence probability by the following formula:

$$P(x) = \prod_{i=1}^{l} P(x_i|x_{1:i-1}) \quad (1)$$

In the training phase, given normal training system call sequence data, this LSTM-based system call language model is trained using the back-propagation through time (BPTT) algorithm. The training criterion minimizes the cross-entropy loss, which is equivalent to maximizing the likelihood of the system call sequence. A standard RNN often suffers from the vanishing/exploding gradient problem, and when training with BPTT, gradient values tend to blow up or vanish exponentially. This makes it difficult to learn long-term dependency in RNNs [23]. LSTM, a well-designed RNN architecture component, is equipped with an explicit memory cell and tends to be more effective to cope with this problem, resulting in numerous successes in recent RNN applications.

When a new query system-call sequence is given, on the assumption that abnormal call patterns deviate from learned normal patterns, yielding significantly lower probabilities than those of normal call patterns, a sequence with an average negative log-likelihood (NLL) above a threshold is classified as abnormal, while a sequence with an average negative log-likelihood below the threshold is classified as normal.

Because typical processes in the system execute a long chain of system calls, the number of system calls required to fully understand the meaning of a system-call sequence is quite large. In addition, the system calls comprising a process are intertwined with each other in a complicated way. The boundaries between system-

call sequences are also vague. In this regard, learning long-term dependence is crucial for devising effective intrusion detection systems.

Markov chains and hidden Markov models are widely used probabilistic models that can estimate the probability of the next call given a sequence of previous calls. There has been previous work on using Markov models in intrusion detection systems [24-27]. However, these methods have an inherent limitation in that the probability of the next call is decided by only a finite number of previous calls. Moreover, LSTM can model exponentially more complex functions than Markov models by using continuous space representations. This property alleviates the data sparsity issue that occurs when a large number of previous states are used in Markov models. In short, the advantages of LSTM models compared to Markov models are two folds: the ability to capture long-term dependency and enhanced expressive power.

2.2. Baseline Classifiers

Deep neural networks are an excellent representation learning method. We exploit the sequence representation learned from the final hidden vector of the LSTM layer after feeding all the sequences of calls. For comparison with our main classifier, we use two baseline classifiers that are commonly used for anomaly detection exploiting vectors corresponding to each sequence: k-nearest neighbor (kNN) and k-means clustering (kMC). Examples of previous work for mapping sequences into vectors of fixed-dimensional hand-crafted features include normalized frequency and tf-idf [28, 29].

Let T be a normal training set, and let $\mathrm{lstm}(x)$ denotes a learned representation of call sequence x from the LSTM layer. kNN classifiers search for k nearest neighbors in T of query sequence x on the embedded space and

measure the minimum radius to cover them all. The minimum radius $g(x;k)$ is used to classify query sequence x. Alternatively, we can count the number of vectors within the fixed radius, $g(x;r)$. In here, we used the former. Because the computational cost of a kNN classifier is proportional to the size of T, using a kNN classifier would be intolerable when the normal training dataset becomes larger.

$$g(x;k) = \min r \quad \text{s.t.} \sum_{y \in T} [d(\text{lstm}(x), \text{lstm}(y)) \leq r] \geq k \tag{2}$$

$$g(x;r) = 1 - \frac{1}{|T|} \sum_{y \in T} [d(\text{lstm}(x), \text{lstm}(y)) \leq r] \tag{3}$$

The kMC algorithm partitions T on the new vector space into k clusters $G_1, G_2, \ldots, G_k$ in which each vector belongs to the cluster with the nearest mean so as to minimize the within-cluster sum of squares. They are computed by Lloyd's algorithm and converge quickly to a local optimum. The minimum distance from each center of clusters μ_i, $h(x;k)$, is used to classify the new query sequence.

$$h(x;k) = \min_{i=1,\cdots,k} d(\text{lstm}(x), \mu_i) \tag{4}$$

The two classifiers C_g and C_h are closely related in that the kMC classifier is equivalent to the 1-nearest neighbor classifier on the set of centers. In both cases of kNN and kMC, we need to choose parameter k empirically, depending on the distribution of vectors. In addition, we need to choose a distance metric on the embedding space; we used the Euclidean distance measure in our experiments.

2.3 Performance Evaluation

If function $f \in S^* \mapsto \mathbb{R}$, which maps a system call sequence to a real value, is given, we can define a thresholding classifier as follows:

$$C_f(x;\theta) = \begin{cases} normal & for f(x) \le \theta \\ abnormal & otherwise. \end{cases} \quad (5)$$

A receiver operating characteristic (ROC) curve can be drawn by changing the threshold value. Commonly, IDS is evaluated by the ROC curve rather than a single point corresponding to a specific threshold on the curve. Sensitivity to the threshold is shown on the curve. The x-axis of the curve represents false alarm rates, and the y-axis of the curve represents detection rates. (A false alarm rate is the ratio of validation normal data classified as abnormal. A detection rate is the ratio of detected attacks in the real attack data.) If the threshold is too low, the IDS is able to detect attacks well, but users would be annoyed due to false alarms. Conversely, if the threshold is too high, false alarm rates becomes lower, but it is easy for IDS to miss attacks. ROC curves closer to $(0,1)$ means a better classifier (i.e., a better intrusion detection system). The area under curve (AUC) summarizes the ROC curve into a single value in the range $[0,1]$ [39].

Most of the intrusion detection algorithms, including our proposed method, employ a thresholding classifier. For the sake of explanation, we define a term 'highly normal' sequence for the classifier C_f as a system call sequence having an extremely low f value so it will be classified as normal even when the threshold θ is sufficiently low to discriminate true abnormals. Highly normal sequences are represented as a flat horizontal line near $(1,1)$ in the ROC curve. The more the classifier finds highly normal sequences, the longer this line is. Note that a highly normal sequence is closely related to the false alarm rate.

Though system call traces themselves might be easy to acquire, collecting or generating a sufficient amount of meaningful traces for the evaluation of intrusion detection systems is a nontrivial task. In order to aid researchers in this regard, the following datasets were made publicly available from prior work: ADFA-LD [30],

KDD98 [31] and UNM [32]. The KDD98 and UNM datasets were released in 1998 and 2004, respectively. Although these two received continued criticism about their applicability to modern systems [33-35], we include them as the results would show how our model fares against early works in the field, which were mostly evaluated on these datasets. As the ADFA-LD dataset was generated around 2012 to reflect contemporary systems and attacks, we have done our evaluation mainly on this dataset.

The ADFA-LD dataset was captured on an x86 machine running Ubuntu 11.04 and consists of three groups: normal training traces, normal validation traces, and attack traces. The KDD98 dataset was audited on a Solaris 2.5.1 server. We processed the audit data into system call traces per session. Each session trace was marked as normal or attack depending on the information provided in the accompanied bsm.list file, which is available alongside the dataset. Among the UNM process set, we tested our model with lpr that was collected from SunOS 4.1.4 machines. We merged the live lpr set and the synthetic lpr set. This combined dataset is further categorized into two groups: normal traces and attack traces. To maintain consistency with ADFA-LD, we divided the normal data of KDD98 and UNM into training and validation data in a ratio of 1:5, which is the ratio of the ADFA-LD dataset. The numbers of system-call sequences in each dataset we used are summarized in Table 1.

Table 1: Summary of datasets used for experiments.

Benchmark	**Normal**		**Attack**	
	# training	# validation	# type	# attack
ADFA-LD	833	4372	6	746
KDD98	1364	5459	10	41
UNM-lpr	627	3136	1	2002

We used ADFA-LD and built three independent system-call language models by changing the hyper-parameters of the LSTM layer: (1) one layer with 200 cells, (2) one layer with 400 cells, and (3) two layers with 400 cells. We matched the number of cells and the dimension of the embedding vector. Our parameters were uniformly initialized in $[-0.1,0.1]$. For computational efficiency, we adjusted all system-call sequences in a mini-batch to be of similar lengths. We used the Adam optimizer [36] for stochastic gradient descent with a learning rate of 0.0001. The normalized gradient was rescaled whenever its norm exceeded 5 [37], and we used dropout [38] with probability 0.5. We show the ROC curves obtained from the experiment in Figure 2.

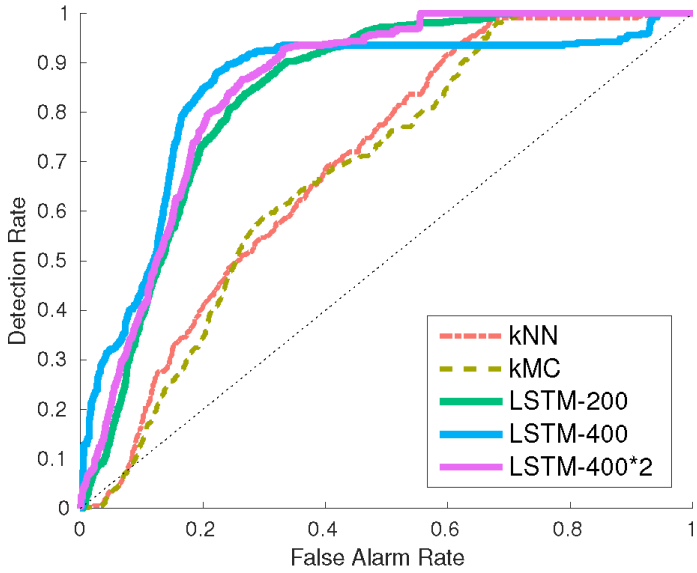

Figure 2: ROC curves of three system-call language models with different parameters and two baseline classifiers from the ADFA-LD.

For the two baseline classifiers, we used the Euclidean distance measure. Changing the distance measure to another metric did not perform well on average.

In case of kNN, using $k = 11$ achieved the best performance empirically. For kMC, using $k = 1$ gave the best performance. Increasing the value of k produced similar but poorer results. We speculate the reason why a single cluster suffices as follows: learned representation vectors of normal training sequence are symmetrically distributed. The kNN classifier C_g and the kMC classifier C_h achieved similar performance. Compared to [28, 29], our baseline classifiers easily returned 'highly normal' calls. This result was leveraged by the better representation obtained from the proposed system-call language modeling. As shown in the Figure 2, three LSTM classifiers performed better than C_g and C_h.

Chapter 3 Ensemble Method to Reduce False Alarms

3.1 Ensemble Method

Building a ‘strong normal’ model (a model representing system-call sequences with high probabilities of being normal) is challenging because of over-fitting issues. In other words, a lower training loss does not necessarily imply better generalization performance. We can consider two reasons for encountering this issue.

First, it is possible that only normal data were used for training the IDS without any attack data. Learning discriminative features that can separate normal call sequences from abnormal sequences is thus hard without seeing any abnormal sequences beforehand. This is a common obstacle for almost every anomaly detection problem. In particular, malicious behaviors are frequently hidden and account for only a small part of all the system call sequences.

Second, in theory, we need a huge amount of data to cover all possible normal patterns to train the model satisfactorily. However, doing so is often impossible in a realistic situation because of the diverse and dynamic nature of system call patterns. Gathering live system-call data is harder than generating synthetic system-call data. The generation of normal training data in an off-line setting can create artifacts, because these data are made in fixed conditions for the sake of convenience in data generation. This setting may cause normal patterns to have some bias.

All these situations make it more difficult to choose a good set of hyper-parameters for LSTM architecture. To cope with this challenge, we propose a new ensemble method. Due to the lack of data, different models with different parameters capture slightly different normal patterns.

Our goal is to minimize the false alarm rate through the composition of multiple classifiers $C_{f_1}, C_{f_2}, \ldots, C_{f_m}$ into a single classifier $C_{\overline{f}}$, resulting in accumulated 'highly normal' data (here m is the number of classifiers used in the ensemble). This is due to the fact that a low false alarm rate is an important requisite in computer security, especially in intrusion detection systems. Our ensemble method can be represented by a simple formula:

$$\overline{f}(x) = \sum_{i=1}^{m} w_i \sigma(f_i(x) - b_i) \tag{6}$$

As activation function σ, we used a leaky ReLU function, namely $\sigma(x) = \max(x, 0.001x)$. Intuitively, the activation function forces potential 'highly normal' sequences having f values lower than b_i to keep their low f values to the final $\overline{f}$ value. If we use the regular ReLU function instead, the degree of 'highly normal' sequences could not be differentiated. We set the bias term b_i to the median of f values of the normal training data. In (6), w_i indicates the importance of each classifier f_i. Because we do not know the performance of each classifier before evaluation, we set w_i to $1/m$. Mathematically, this appears to be a degenerated version of a one-layer neural network. The basic philosophy of the ensemble method is that when the classification results from various classifiers are slightly different, we can make a better decision by composing them well. Still, including bad classifiers could degrade the overall performance. By choosing classifiers carefully, we can achieve satisfactory results in practice, as will be shown in Section 3.2.

3.2 Comparison with Other Methods

We assume that the three LSTM classifiers we trained are strong enough by themselves, and their classification results would be different from each other. By

applying ensemble methods, we would expect to improve the performance. The first one was averaging, the second one was voting, and lastly we used our ensemble method as we explained in Section 3.1. The proposed ensemble method gave a better AUC value (0.928) with a large margin than that of the averaging ensemble method (0.890) and the voting ensemble method (0.859). Moreover, the curve obtained from the proposed ensemble method was placed above individual single curves, while other ensemble methods did not show this property.

In the setting of anomaly detection where attack data are unavailable, learning ensemble parameters is infeasible. If we exploit partial attack data, the assumption breaks down and the zero-day attack issue remains. Our ensemble method is appealing in that it performs remarkably well without learning.

To be clear, we applied ensemble methods to three LSTM classifiers learned independently using different hyper-parameters, not with the baseline classifiers, C_g or C_h. Applying ensemble methods to each type of baseline classifier gave unsatisfactory results since changing parameters or initialization did not result in complementary and reasonable classifiers that were essential for ensemble methods. Alternatively, we could do ensemble our LSTM classifiers and baseline classifiers together. However, this would also be a wrong idea because their f values differ in scale. The value of f in our LSTM classifier is an average negative log-likelihood, whereas g and h indicate distances in a continuous space.

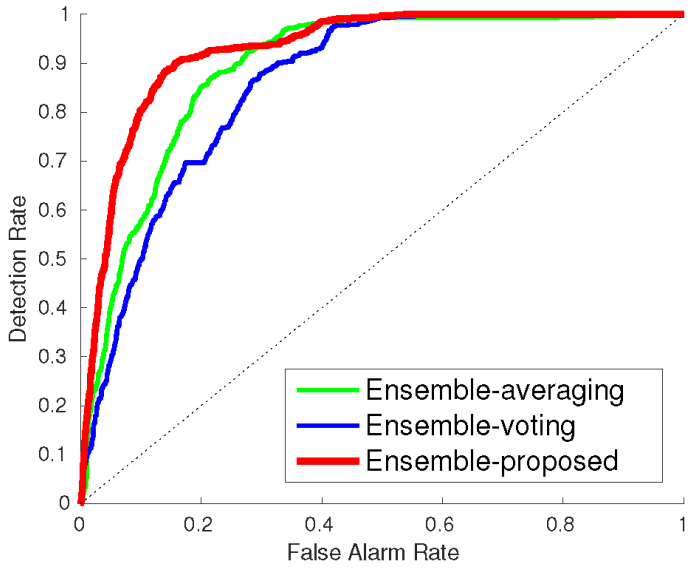

Figure 3: ROC curves of different ensemble methods from the ADFA-LD.

According to [4], the extreme learning machine (ELM) model, sequence time-delay embedding (STIDE), and the hidden Markov model (HMM) [3, 40] achieved about 13%, 23%, and 42% false alarm rates (FAR) for 90% detection rate (DR), respectively. We achieved 16% FAR for 90% DR, which is comparable to the result of ELM and outperforms those of STIDE and HMM. The ROC curves for ELM, HMM, and STIDE can be found, but we could not draw those curves on the same plot with ours because the authors provided no specific details of their results. [4] classified ELM as a semantic approach and other two as syntactic approaches which treat each call as a basic unit. To be fair, our proposed method should be compared with those approaches that use system calls only as a basic unit in that we watch the sequence call-by-call. Furthermore, our method is end-to-end while ELM relies on hand-crafted features.

In [4], the authors reported that there was significant overhead for training the models mentioned above, and the overhead would inevitably increase for handling larger data. Longer phrases tend to be more informative, but handling them typically requires larger dictionaries. For this reason, [4] had to put an empirical

upper bound to limit the lengths of phrases, which then might lower the performance of the models to handle various attacks. By contrast, our approach can learn in continuous space semantically meaningful representations of calls, phrases, and sequences of arbitrary lengths. Moreover, our method can relieve the burden of preprocessing (potentially massive) logging data. We expect that incorporating prior knowledge into our model can further boost its performance.

Chapter 4 Interpretation to Transfer Learning

4.1 Portability

The experiments similar to those presented in Section 2.3 using the KDD98 dataset and the UNM dataset were carried out. First, we a system-call language model with LSTM having one layer of 200 cells and a classifier are trained using the normal training traces of the KDD98 dataset. The same model was used to evaluate the UNM dataset to examine the portability of the LSTM models trained with data from a different but similar system. The results of the experiments are represented in Figure 4. For comparison, we display the ROC curve of the UNM dataset by using the model from training the normal traces therein. To examine portability, the system calls in test datasets need to be included or matched to those of training datasets. UNM was generated using an earlier version of OS than that of KDD98, but ADFA-LD was audited on a fairly different OS. This made our experiments with other combinations difficult.

Through a quantitative analysis, for the KDD98 dataset, we earned an almost perfect ROC curve with an AUC value of 0.994 and achieved 2.3% FAR for 100% DR. With the same model, we tested the UNM datset and obtained a ROC curve with an AUC value of 0.969 and 5.5% FAR for 99.8% DR. This result was close to the result earned by using the model trained on normal training traces of the UNM dataset itself, as shown in the right plot of Figure 4.

This result is intriguing because it indicates that system-call language models have a strong portability. In other words, after training one robust and extensive model, the model can then be deployed to other similar host systems. By doing so, we can mitigate the burden of training cost. This paradigm is closely related to the concept of transfer learning, or zero-shot learning. It is well known that neural

networks can learn abstract features and that they can be used successfully for unseen data. For example, trained neural networks for image classification can serve as a feature extractor or they can be fine-tuned for different kinds of image groups.

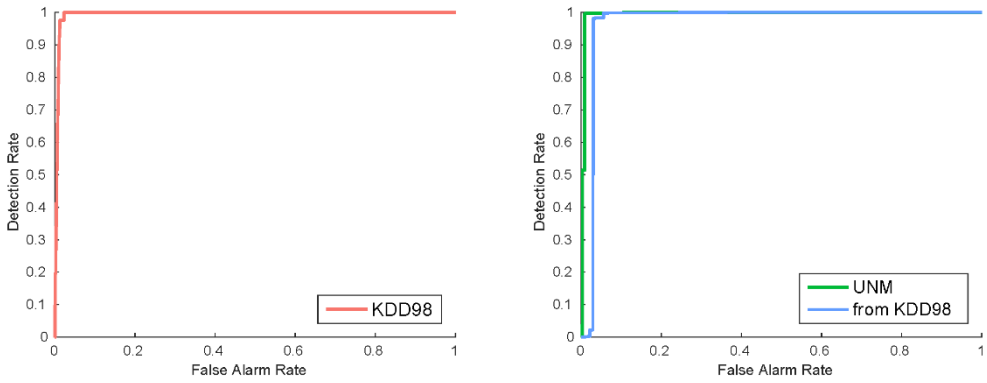

Figure 3: ROC curves from the KDD dataset and UNM dataset. Left is the evaluation about the KDD dataset. Right is the evaluation about UNM dataset using the model trained with the KDD98 dataset and the UNM dataset.

4.2 Visualization of Learned Representations

It is well-known that neural network based-language models can learn semantically meaningful embeddings to continuous space [6, 41, 42]. We expected to see a similar characteristic with the proposed system-call language model. The 2D projection of the calls using the embedding matrix W learned from the system-call language model was done by t-SNE [43] and shown in Figure 5. Just as the natural language model, we can expect that calls having similar co-occurrence patterns are positioned in similar locations in the embedded space after training the system call language model. We can clearly see that calls having alike functionality are clustered with each other.

The first obvious cluster would be the *read-write* call pair and the *open-close*

pair. The calls of each pair were located in close proximity in the space, meaning that our model learned to associate them together. At the same time, the difference between the calls of each pair appears to be almost the same in the space, which in turn would mean our model learned that the relationship of each pair somewhat resembles [42].

Another notable cluster would be the group of *select*, *pselect6*, *ppoll*, *epoll_wait* and *nanosleep*. The calls *select*, *pselect6* and *ppoll* all have nearly identical functions in that they wait for some file descriptors to become ready for some class of I/O operation or for signals. The other two calls also have similar characteristics in that they wait for a certain event or signal as well. This could be interpreted as our model learning that these 'waiting' calls share similar characteristics.

Other interesting groups would be: *readlink* and *lstat64* which are calls related to symbolic links; *fstatat64* and *fstat64* which are calls related to stat calls using file descriptors; *pipe* and *pipe2* which are nearly identical and appear almost as one on the embedding layer. These cases show that our model is capable of learning similar characteristics among the great many system calls.

We believe that this learned representation of calls can be used for further tasks exploiting neural networks and system calls as a basic unit. For better representation, computationally efficient predictive model such as continous bag-of-words (CBOW) and the skip-gram can be adapted [44]. Negative sampling method [45] which enable much faster training will assign high probability to the real calls and low probabilities to noise calls [46].

Similarly to the call representations, we expected that attack sequences with the same type would cluster to each other, and we tried to visualize them. However, for various reasons including the lack of data, we were not able to observe this

phenomenon. Taking the fact that detecting abnormal patterns from normal patterns well would be sufficiently hard into consideration, learning representation to separate different abnormal patterns with only seen normal patterns would also be an extremely difficult task.

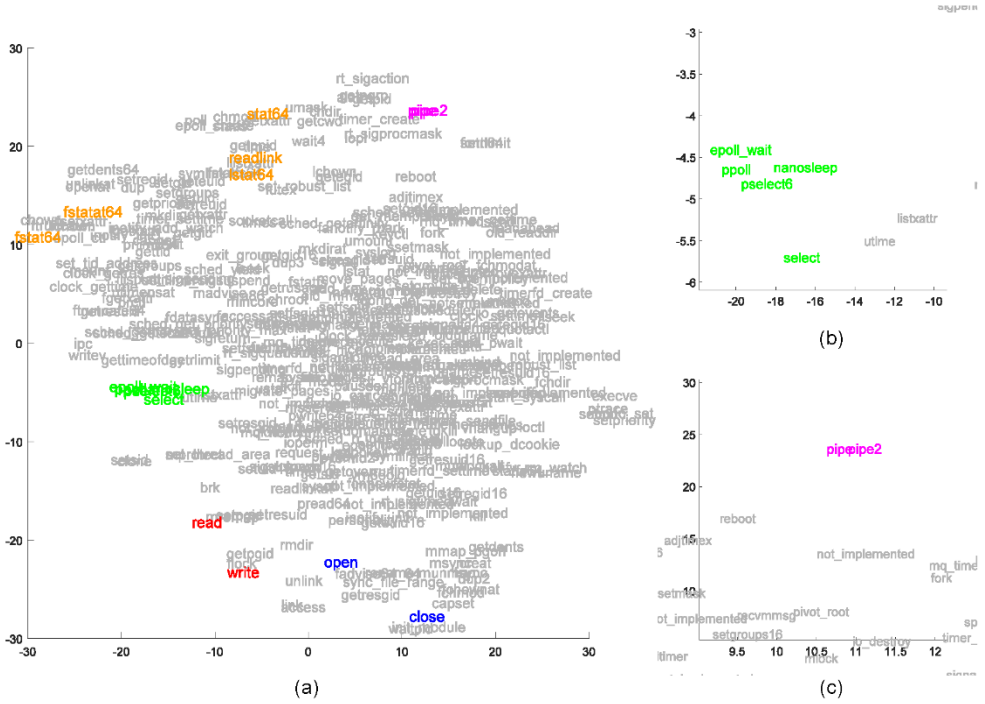

Figure 4: 2-D embedding of learned call representations. (a) shows the full representation space of system calls that appeared in training data. (b) and (c) show the zoomed-in view of specific regions.

Chapter 5 Generalization to Branch Sequences

System calls are special kinds of branches. Attackers who want to infiltrate a system usually gain control over program execution exploiting exposed vulnerabilities and resultantly divert the execution flow at their disposal, which program to behave deviantly. To handle many cases where only call sequences are not sufficient or accurate enough to reflect true behaviors of a program, we extend basic tokens of sequences to branches (including all system and function calls) which can characterize the entire execution path of a program rather than traditional calls. Factually, contemporary attacks are executed in more fine-grained level and hard to be detected by only looking at system calls.

By the way, this generalization causes some problems for the machine learning system. First, branch sequences are much longer than system calls when comparing things recorded at the same time. Second, varied addresses of possible branches make harder to predict the next branch. In practice, branch never appeared in training process could occur but we can not sure it is anomalous. In this chapter, I will explain how to solve those problems and describe their experimental results.

5.1 Handling Open Vocabulary Problem

In general, it is hard to know all possible branch target addresses beforehand and they are not fixed. To deal with this issue, we build a vocabulary that consists of the top $K-1$ most frequent addresses (or addresses appeared at least certain amount of times in training data) and a single special address, *unknown*, which indicates any other address. The maximum size of a vocabulary or the minimum

number of occurrence to be included in a vocabulary is determined empirically. After this preprocessing, the remaining procedures are the same as the close vocabulary situation.

5.2 Experiments on Branch Sequences

Different with system call sequences, open datasets of branch sequences with labels are not available. Therefore, we generated new datasets by logging all branch sequences while several programs are running and used them for training and evaluation. For logging branches, Processor Trace (PT) [47] is used. PT is Intel's debug tracing unit and has been installed in the latest versions of x86/x64 architectures starting from Broadwell.

Branch language models are trained on three individual real programs: MySQL, Nagios and ProFTPD [48-50]. They were the target victims of three publicly available attacks: privilege escalation via CVE-2016-6663 and CVE-2016-6664 [51, 52], privilege escalation via CVE-2016-9565 and CVE-2016-9566 [53, 54] and data-oriented programming (DOP) attack via CVE-2006-5815 [55].

PT collects the branch sequences of all the processes relevant to these programs. The recording environment for normal branch sequences is controlled to guarantee that no attacks or suspicious actions are executed. In the case of MySQL and ProFTPD, a mix of human input, statements or commands to be exact, and simulated human input, a random combination of records of actual human input activities is provided. In the case of Nagios, three actively using desktop computers are monitored. Attack data are generated by repeating instruction manuals.

Statistics of generated datasets including the number of long sequences and the total number of all branches per each programs' dataset are illustrated in Table 2. We separate normal data into the training set and the validation set. We have constructed the vocabulary per programs as the set of all branches that appear at least

10 times in the training data. Including branches appeared too few times could lead to overestimation of their likelihood.

Table 2: Summary of generated datasets.

Program	**Normal**			**Attack**
	# training	K	# validation	# attack
MySQL	(4; 28,511,573)	7990	(20; 370,815,414)	(40; 223,879,404)
Nagios	(4; 9,072,911)	6445	(15; 88,772,864)	(18; 293,738,070)
ProFTPD	(4; 44,668,630)	2759	(9; 107,664,903)	(10; 100,479,122)

We evaluate the branch language models with one LSTM layer of 200 cells. Figure 6 depicts the perplexity of sequences collected for evaluation in log scale. This shows our branch language models are able to discern between the normal and attack sequences. The perplexity for the normal sequences are quite low. On the other hand, nearly all sequences containing attacks show high perplexity, which would mean the branch sequences deviate from what the model expected. Though the threshold of classifiers can be changed for desired accuracy or false alarm rates, we set the threshold as a 99-percentile of negative log-likelihood probability at every branch in the training data for our evaluation. They are depicted as a red line in the Figure 6. With this threshold, all attack sequences are detected (100% DR) and for MySQL, Nagios, ProFTPD programs, 2, 4, 1 normal sequences are classified as abnormal resulting in 4.8%, 1.8%, and 9.1% FAR, respectively.

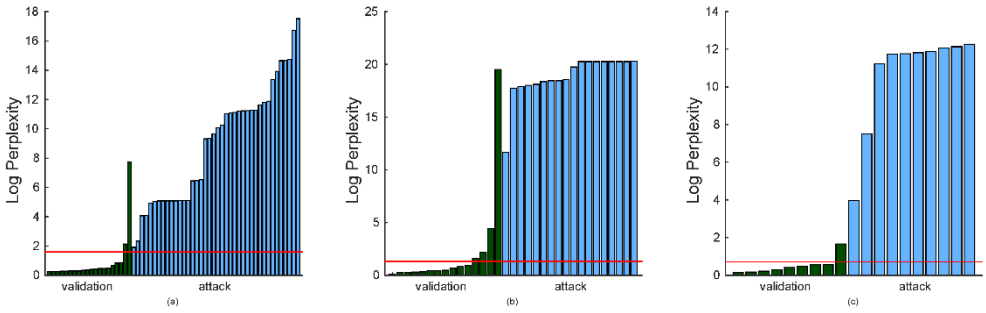

Figure 5: The perplexity of normal sequences and attack sequences. (a), (b), (c) corresponds to MySQL, Nagios, and ProFTPD, respectively. Red horizontal lines on each graph represent threshold configured from the training data

For a few normal sequences, our model reports high perplexity. With further examination, we believe this is due to the rare events, such as one of the Nagios monitored computers crashing that were not covered by the training data. This shows that our model can successfully interpret branch sequences of such events as an anomaly in program behavior. If we want to learn all legitimate patterns, techniques for test input generation [56] could be incorporated.

5.3 Discussion on Branch Language Model

We have shown that our approach is applicable to three types of applications and corresponding attacks but more extensive verification on various applications and attacks are required. Also, we need to investigate the comparison of the two methods using calls and branch.

If we consider a maximum likelihood training objective and LSTM architecture is complex enough to model patterns of branch sequences, overfitting can occur. In that case, the model will give probability close to zero for unknown branch resulting in extremely high negative log-likelihood. Then, a classifier may

act just similarly with an unknown branch detector. Smoothing technique [57, 58] for unknown branches can lessen this phenomenon. However, how much we permit unknown branches is an open question and depends on the purpose.

It is questionable that using branch addresses as the unit of our language model is adequate. In my opinion, more important thing is which code is actually executed. Branch at very close but different address would be regarded as completely different while they are related. One possible way to represent branches better is mapping a representation of each branch to a representation of code sequences when that branch operation occurred. It is similar to a char-based encoding of words in natural language processing. This transformation can be done for all branches or for only unknown branches [59, 60].

Chapter 6 Future Work

The main purpose of this work was to check the applicability of our LSTM language modeling approach for intrusion detection systems rather than exploring the limits of its performance. To boost the accuracy and make possible real applications, there are more ways to go. In this chapter, I will talk about future work briefly.

6.1 Advanced Model Architecture

Word-level RNNs were most common for sequential data, especially NLP and had shown outstanding performance. There are diverse streams of variations using neural networks nowadays, like changing model architecture or the basic unit fed into neural networks [61]. While recurrent neural networks are the most standard way to deal with sequential data, Convolutional Neural Networks (CNNs) used for sequential data also have shown great performance, even sometimes better than RNN [62-65]. Convolutional generative models such as PixcelCNN [66, 67] can be used for our language model. Multi-layer CNNs allows to learn hierarchical representations. Gate units and attention steps could help to focus on important regions.

Also, we anticipate that a hybrid method that combines signature-based and anomaly-based approaches will allow us to create more accurate intrusion detection systems.

6.2 Finding Anomalous Segments

Previously, the decision whether given sequence is normal or abnormal was based on the entire sequence. However, anomalous behaviors correspond to a very small portion of the whole sequence and they are often hidden on purpose, for example, mimicry attacks [68, 69]. It makes anomaly detection tasks more challenging.

Program execution consists of a long chain of branches. For the classification, we need to split sequences arbitrary but there is no explicit boundary and no justification for arbitrary segmentation. Besides, IDS may be not only curious about whether an anomalous behavior was detected but also when and where the behavior is executed for further examination and prevention. Therefore, the ability to pinpoint candidates of exact anomalous segments in the entire sequence is required for real intrusion detection systems.

It is possible to determine if there exists a sufficiently long section (rather than the whole sequence) with the average NLL above the threshold. In simple terms, a maximal segment of average NLL is a segment with a single element of maximum NLL. To achieve meaningful results, the minimum length of the segment should be fixed. Finding segments with maximal average NLL and longer than certain lower-bound can be done in linear time [70] meaning that it does not increase the inference time significantly with the help of a parallel computation.

6.3 Adversarial Training

Adversarial examples [71] are inputs to machine learning models that an attacker has intentionally designed to cause the model to output an incorrect answer. They yield small perturbations to original data [72]. Adversarial Training is a simple solution where simply generated adversarial examples are explicitly used to train the

model not to be fooled by each of them. Defending against adversarial examples is difficult in that machine learning models are required to produce correct outputs for all many possible inputs. This is very active research area now [73, 74].

Unlike continuous data such as images, adding small noise to discrete sequences could be critical. Small perturbation on sequential data can result in totally different one. In the same line, while system call or branch sequences are recorded from program behaviors, a mapping from a sequence to program behaviors is not always present. It means our model have some sort of inherent resistance to such attacks.

6.4 Online Learning Framework

Lastly, we are considering designing a new framework to build a robust model in on-line settings by collecting large-scale data generated from distributed environments. As data accumulates as time goes by, batch learning on the entire training data set is infeasible. Effective management of data for training is necessary. We need to care about our language model to keep prior normal patterns which could be overridden by new input sequences.

Chapter 7 Conclusion

Our main contributions for designing intrusion detection systems as described in this thesis have two parts: the introduction of a language modeling approach for system call and branch sequences and a new ensemble method. To the best of my knowledge, our method is the first to introduce the concept of a language model, especially using LSTM, to anomaly-based IDS. The system-call language model can capture the semantic meaning of each call and its relation to other system calls. Moreover, we proposed an innovative and simple ensemble method that can better fit to IDS design by focusing on lowering false alarm rates. We showed its outstanding performance by comparing it with existing state-of-the-art methods and demonstrated its robustness and generality by experiments on diverse benchmarks.

As discussed earlier, the proposed method also has excellent portability. In contrast to alternative methods, our proposed method incurs significant smaller training overhead because it does not need to build databases or dictionaries to keep a potentially exponential amount of patterns. Our method is compact and light in that the size of the space required to save parameters is small. The overall training and inference processes are also efficient and fast, as our methods can be implemented using efficient sequential matrix multiplications.

We generalized a system call language model to a branch language model. The problems that have arisen and the solutions are suggested. Experimental results in some cases are described. However, more careful analysis and validation should be supported to be applied to a real intrusion detection system.

References

1 Jyothsna, V., Prasad, V.R., and Prasad, K.M.: 'A review of anomaly based intrusion detection systems', International Journal of Computer Applications, 2011, 28, (7), pp. 26-35

2 Forrest, S., Hofmeyr, S.A., Somayaji, A., and Longstaff, T.A.: 'A sense of self for unix processes', in Editor (Ed.)^(Eds.): 'Book A sense of self for unix processes' (IEEE, 1996, edn.), pp. 120-128

3 Forrest, S., Hofmeyr, S., and Somayaji, A.: 'The evolution of system-call monitoring', in Editor (Ed.)^(Eds.): 'Book The evolution of system-call monitoring' (IEEE, 2008, edn.), pp. 418-430

4 Creech, G., and Hu, J.: 'A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns', IEEE Transactions on Computers, 2014, 63, (4), pp. 807-819

5 Bahdanau, D., Cho, K., and Bengio, Y.: 'Neural machine translation by jointly learning to align and translate', arXiv preprint arXiv:1409.0473, 2014

6 Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y.: 'Learning phrase representations using RNN encoder-decoder for statistical machine translation', arXiv preprint arXiv:1406.1078, 2014

7 Graves, A., Mohamed, A.-r., and Hinton, G.: 'Speech recognition with deep recurrent neural networks', in Editor (Ed.)^(Eds.): 'Book Speech recognition with deep recurrent neural networks' (IEEE, 2013, edn.), pp. 6645-6649

8 Hermann, K.M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P.: 'Teaching machines to read and comprehend', in Editor (Ed.)^(Eds.): 'Book Teaching machines to read and comprehend' (2015, edn.), pp. 1693-1701

9 Rush, A.M., Chopra, S., and Weston, J.: 'A neural attention model for abstractive sentence summarization', arXiv preprint arXiv:1509.00685, 2015

10 Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y.: 'Exploring the limits of language modeling', arXiv preprint arXiv:1602.02410, 2016

11 Zaremba, W., Sutskever, I., and Vinyals, O.: 'Recurrent neural network regularization', arXiv preprint arXiv:1409.2329, 2014

12 Debar, H., Becker, M., and Siboni, D.: 'A neural network component for an intrusion detection system', in Editor (Ed.)^(Eds.): 'Book A neural network component for an intrusion detection system' (IEEE, 1992, edn.), pp. 240-250

13 Mukkamala, S., Janoski, G., and Sung, A.: 'Intrusion detection using neural networks and support vector machines', in Editor (Ed.)^(Eds.): 'Book Intrusion detection using neural networks and support vector machines' (IEEE, 2002, edn.), pp. 1702-1707

14 Ryan, J., Lin, M.-J., and Miikkulainen, R.: 'Intrusion detection with neural networks', Advances in neural information processing systems, 1998, pp. 943-949

15 Wang, G., Hao, J., Ma, J., and Huang, L.: 'A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering', Expert Systems with

Applications, 2010, 37, (9), pp. 6225-6232
16 Staudemeyer, R.C.: 'Applying long short-term memory recurrent neural networks to intrusion detection', South African Computer Journal, 2015, 56, (1), pp. 136-154
17 Staudemeyer, R.C., and Omlin, C.W.: 'Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data', in Editor (Ed.)^(Eds.): 'Book Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data' (ACM, 2013, edn.), pp. 218-224
18 Canzanese, R., Mancoridis, S., and Kam, M.: 'System call-based detection of malicious processes', in Editor (Ed.)^(Eds.): 'Book System call-based detection of malicious processes' (IEEE, 2015, edn.), pp. 119-124
19 Feng, H.H., Kolesnikov, O.M., Fogla, P., Lee, W., and Gong, W.: 'Anomaly detection using call stack information', in Editor (Ed.)^(Eds.): 'Book Anomaly detection using call stack information' (IEEE, 2003, edn.), pp. 62-75
20 Sekar, R., Bendre, M., Dhurjati, D., and Bollineni, P.: 'A fast automaton-based method for detecting anomalous program behaviors', in Editor (Ed.)^(Eds.): 'Book A fast automaton-based method for detecting anomalous program behaviors' (IEEE, 2001, edn.), pp. 144-155
21 Hochreiter, S., and Schmidhuber, J.: 'Long short-term memory', Neural computation, 1997, 9, (8), pp. 1735-1780
22 Maas, A.L., Hannun, A.Y., and Ng, A.Y.: 'Rectifier nonlinearities improve neural network acoustic models', in Editor (Ed.)^(Eds.): 'Book Rectifier nonlinearities improve neural network acoustic models' (2013, edn.), pp.
23 Bengio, Y., Simard, P., and Frasconi, P.: 'Learning long-term dependencies with gradient descent is difficult', IEEE transactions on neural networks, 1994, 5, (2), pp. 157-166
24 Hoang, X.D., Hu, J., and Bertok, P.: 'A multi-layer model for anomaly intrusion detection using program sequences of system calls', in Editor (Ed.)^(Eds.): 'Book A multi-layer model for anomaly intrusion detection using program sequences of system calls' (Citeseer, 2003, edn.), pp.
25 Hofmeyr, S.A., Forrest, S., and Somayaji, A.: 'Intrusion detection using sequences of system calls', Journal of computer security, 1998, 6, (3), pp. 151-180
26 Hu, J., Yu, X., Qiu, D., and Chen, H.-H.: 'A simple and efficient hidden Markov model scheme for host-based anomaly intrusion detection', IEEE network, 2009, 23, (1), pp. 42-47
27 Yolacan, E.N., Dy, J.G., and Kaeli, D.R.: 'System call anomaly detection using multi-hmms', in Editor (Ed.)^(Eds.): 'Book System call anomaly detection using multi-hmms' (IEEE, 2014, edn.), pp. 25-30
28 Liao, Y., and Vemuri, V.R.: 'Using text categorization techniques for intrusion detection', in Editor (Ed.)^(Eds.): 'Book Using text categorization techniques for intrusion detection' (2002, edn.), pp. 51-59
29 Xie, M., Hu, J., Yu, X., and Chang, E.: 'Evaluating host-based anomaly detection systems: Application of the frequency-based algorithms to adfa-ld', in Editor (Ed.)^(Eds.): 'Book Evaluating host-based anomaly detection systems: Application of the frequency-based algorithms to adfa-ld' (Springer, 2014, edn.), pp. 542-549

30 Creech, G., and Hu, J.: 'Generation of a new IDS test dataset: Time to retire the KDD collection', in Editor (Ed.)^(Eds.): 'Book Generation of a new IDS test dataset: Time to retire the KDD collection' (IEEE, 2013, edn.), pp. 4487-4492
31 Lippmann, R.P., Fried, D.J., Graf, I., Haines, J.W., Kendall, K.R., McClung, D., Weber, D., Webster, S.E., Wyschogrod, D., and Cunningham, R.K.: 'Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation', in Editor (Ed.)^(Eds.): 'Book Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation' (IEEE, 2000, edn.), pp. 12-26
32 http://www.cs.unm.edu/~immsec/systemcalls.htm
33 Brown, C., Cowperthwaite, A., Hijazi, A., and Somayaji, A.: 'Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict', in Editor (Ed.)^(Eds.): 'Book Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict' (IEEE, 2009, edn.), pp. 1-7
34 McHugh, J.: 'Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory', ACM Transactions on Information and System Security (TISSEC), 2000, 3, (4), pp. 262-294
35 Tan, K.M., and Maxion, R.A.: 'Determining the operational limits of an anomaly-based intrusion detector', IEEE Journal on selected areas in communications, 2003, 21, (1), pp. 96-110
36 Kingma, D., and Ba, J.: 'Adam: A method for stochastic optimization', arXiv preprint arXiv:1412.6980, 2014
37 Pascanu, R., Mikolov, T., and Bengio, Y.: 'On the difficulty of training recurrent neural networks', ICML (3), 2013, 28, pp. 1310-1318
38Srivastava, 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, 2014, 15, (1), pp. 1929-1958
39 Bradley, A.P.: 'The use of the area under the ROC curve in the evaluation of machine learning algorithms', Pattern recognition, 1997, 30, (7), pp. 1145-1159
40 Warrender, C., Forrest, S., and Pearlmutter, B.: 'Detecting intrusions using system calls: Alternative data models', in Editor (Ed.)^(Eds.): 'Book Detecting intrusions using system calls: Alternative data models' (IEEE, 1999, edn.), pp. 133-145
41 Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C.: 'A neural probabilistic language model', Journal of machine learning research, 2003, 3, (Feb), pp. 1137-1155
42 Mikolov, T., Yih, W.-t., and Zweig, G.: 'Linguistic Regularities in Continuous Space Word Representations', in Editor (Ed.)^(Eds.): 'Book Linguistic Regularities in Continuous Space Word Representations' (2013, edn.), pp. 746-751
43 Maaten, L.v.d., and Hinton, G.: 'Visualizing data using t-SNE', Journal of Machine Learning Research, 2008, 9, (Nov), pp. 2579-2605
44 Mikolov, T., Chen, K., Corrado, G., and Dean, J.: 'Efficient estimation of word representations in vector space', arXiv preprint arXiv:1301.3781, 2013
45 Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., and Dean, J.: 'Distributed representations of words and phrases and their compositionality', in Editor (Ed.)^(Eds.): 'Book Distributed representations of words and phrases and their

compositionality' (2013, edn.), pp. 3111-3119
46 Mnih, A., and Kavukcuoglu, K.: 'Learning word embeddings efficiently with noise-contrastive estimation', in Editor (Ed.)^(Eds.): 'Book Learning word embeddings efficiently with noise-contrastive estimation' (2013, edn.), pp. 2265-2273
47 https://software.intel.com/en-us/articles/intel-sdm
48 https://www.nagios.org/
49 https://www.mysql.com/
50 http://www.proftpd.org/
51 https://www.exploit-db.com/exploits/40678/
52 https://www.exploit-db.com/exploits/40679/
53 https://www.exploit-db.com/exploits/40920/
54 https://www.exploit-db.com/exploits/40921/
55 Hu, H., Shinde, S., Adrian, S., Chua, Z.L., Saxena, P., and Liang, Z.: 'Data-oriented programming: On the expressiveness of non-control data attacks', in Editor (Ed.)^(Eds.): 'Book Data-oriented programming: On the expressiveness of non-control data attacks' (IEEE, 2016, edn.), pp. 969-986
56 Cadar, C., Dunbar, D., and Engler, D.R.: 'KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs', in Editor (Ed.)^(Eds.): 'Book KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs' (2008, edn.), pp. 209-224
57 Chen, S.F., and Goodman, J.: 'An empirical study of smoothing techniques for language modeling', in Editor (Ed.)^(Eds.): 'Book An empirical study of smoothing techniques for language modeling' (Association for Computational Linguistics, 1996, edn.), pp. 310-318
58 Xie, Z., Wang, S.I., Li, J., Lévy, D., Nie, A., Jurafsky, D., and Ng, A.Y.: 'Data Noising as Smoothing in Neural Network Language Models', arXiv preprint arXiv:1703.02573, 2017
59 Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T.: 'Enriching word vectors with subword information', arXiv preprint arXiv:1607.04606, 2016
60 Luong, M.-T., and Manning, C.D.: 'Achieving open vocabulary neural machine translation with hybrid word-character models', arXiv preprint arXiv:1604.00788, 2016
61 Kim, Y., Jernite, Y., Sontag, D., and Rush, A.M.: 'Character-aware neural language models', in Editor (Ed.)^(Eds.): 'Book Character-aware neural language models' (2016, edn.), pp.
62 Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P.: 'Natural language processing (almost) from scratch', Journal of Machine Learning Research, 2011, 12, (Aug), pp. 2493-2537
63 Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y.N.: 'Convolutional Sequence to Sequence Learning', arXiv preprint arXiv:1705.03122, 2017
64 Kalchbrenner, N., Grefenstette, E., and Blunsom, P.: 'A convolutional neural network for modelling sentences', arXiv preprint arXiv:1404.2188, 2014
65 Kim, Y.: 'Convolutional neural networks for sentence classification', arXiv preprint arXiv:1408.5882, 2014
66 Oord, A.v.d., Kalchbrenner, N., and Kavukcuoglu, K.: 'Pixel recurrent neural networks', arXiv preprint arXiv:1601.06759, 2016
67 van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., and Graves, A.:

‘Conditional image generation with pixelcnn decoders’, in Editor (Ed.)^(Eds.): ‘Book Conditional image generation with pixelcnn decoders’ (2016, edn.), pp. 4790-4798
68 Shu, X., Yao, D., and Ramakrishnan, N.: ‘Unearthing stealthy program attacks buried in extremely long execution paths’, in Editor (Ed.)^(Eds.): ‘Book Unearthing stealthy program attacks buried in extremely long execution paths’ (ACM, 2015, edn.), pp. 401-413
69 Wagner, D., and Soto, P.: ‘Mimicry attacks on host-based intrusion detection systems’, in Editor (Ed.)^(Eds.): ‘Book Mimicry attacks on host-based intrusion detection systems’ (ACM, 2002, edn.), pp. 255-264
70 Chung, K.-m., and Lu, H.-I.: ‘An optimal algorithm for the maximum-density segment problem’, SIAM Journal on Computing, 2005, 34, (2), pp. 373-387
71 Goodfellow, I.J., Shlens, J., and Szegedy, C.: ‘Explaining and harnessing adversarial examples’, arXiv preprint arXiv:1412.6572, 2014
72 Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R.: ‘Intriguing properties of neural networks’, arXiv preprint arXiv:1312.6199, 2013
73 Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., and Swami, A.: ‘Practical black-box attacks against machine learning’, in Editor (Ed.)^(Eds.): ‘Book Practical black-box attacks against machine learning’ (ACM, 2017, edn.), pp. 506-519
74 Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A.: ‘Distillation as a defense to adversarial perturbations against deep neural networks’, in Editor (Ed.)^(Eds.): ‘Book Distillation as a defense to adversarial perturbations against deep neural networks’ (IEEE, 2016, edn.), pp. 582-597

Abstract

LSTM Language Modeling for Intrusion Detection Systems

Gyuwan Kim

Dept. of Electrical and Computer Engineering

The Graduate School

Seoul National University

In computer security, designing a robust intrusion detection system is one of the most fundamental and important problems. In this thesis, we propose a language modeling approach to system call sequences and branch sequences for designing anomaly-based host intrusion detection systems. To remedy the issue of high false-alarm rates commonly arising in conventional methods, we employ a novel ensemble method that blends multiple thresholding classifiers into a single one, making it possible to accumulate 'highly normal' sequences. The proposed system call language model has various advantages leveraged by the fact that it can learn the semantic meaning and interactions of each system call that existing methods cannot effectively consider. Through diverse experiments on public benchmark datasets and generated datasets, we demonstrate the validity and effectiveness of the proposed method. Moreover, we show that our model possesses high portability, which is one of the key aspects of realizing successful intrusion detection systems.

Keywords : Computer Security, Language Model, Anomaly Detection, LSTM, Deep Learning, Ensemble Method (6단어 이내)

Student Number : 2015-20891