MASKED: A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System

Md. Shariful Islam, Korosh Koochekian Sabor, Wahab Hamou-Lhadj, Abdelaziz Trabelsi
Department of Electrical and Computer Engineering
Concordia University, Montreal, QC, Canada
{mdsha_i,k_kooche,abdelw,trabelsi}@ece.concordia.ca

Luay Alawneh
Department of Software Engineering
Jordan University of Science and Technology, Irbid, Jordan
lmalawneh@just.edu.jo

Abstract— **Detecting system anomalies at run-time is critical for system reliability and security. Studies in this area focused mainly on effectiveness of the proposed approaches; that is, the ability to detect anomalies with high accuracy. However, less attention was given to efficiency. In this paper, we propose an efficient MapReduce Solution for the Kappa-pruned Ensemble based Anomaly Detection System (MASKED). It profiles the heterogeneous features from large-scale traces of system calls and processes them by heterogeneous anomaly detectors which are Sequence-Time Delay Embedding (STIDE), Hidden Markov Model (HMM), and One-class Support Vector Machine (OCSVM). We deployed MASKED on a Hadoop cluster using the MapReduce programming model. We compared their efficiency and scalability by varying the size of the cluster. We assessed the performance of the proposed approach using the CANALI-WD dataset which consists of 180 GB of execution traces, collected from 10 different machines. Experimental results show that MASKED becomes more efficient and scalable as the file size is increased (e.g., 6-node cluster is 8 times faster than the 2-node cluster). Moreover, the throughput achieved on a 6-node solution is up to 5 times better than a 2-node solution.**

Keywords- Anomaly Detection, Boolean Combination, Ensemble, Heterogeneous Detectors, Hadoop, MapReduce, System Call Traces.

I. INTRODUCTION

Anomaly detection refers to the problem of finding unexpected patterns of system or user generated data that do not conform to the normal behavior. Anomaly detection is used in a wide variety of applications such as fraud detection for credit cards, insurance or health care, intrusion detection for cyber-security, etc. In security, which is the main focus of this paper, system anomalies may occur due to attacks caused by intruders. Detecting system anomalies is therefore an important task that can enhance system reliability.

The last two decades have seen an increase in attention to the field of anomaly detection. Several approaches have emerged using panoply of methods including statistical methods, machine learning, and data mining [14][30][33][45]. Although these techniques vary in their design, their common objective was to build a model that represents the normal behavior of a system, which can then be used to detect deviations from normalcy. Most anomaly detection techniques use the temporal order of system calls, generated by a process at the kernel level, as features [7][32][33][42].

Studies have shown that ensemble approaches that combine the decisions of multiple crisp detectors [1] using Boolean combination rules such as Pair-wise Brute-force Boolean Combination (BBC2) [27], Iterative Boolean Combination (IBC) [42], and a recently proposed Weighted Pruning Iterative Boolean Combination (WPIBC) [28], greatly improve the detection accuracy, while reducing the false alarm rates which are a major impediment for the general adoption of anomaly detection techniques in practice. Moreover, Wael et al. [44] have shown that a combination of heterogeneous anomaly detectors (e.g., STIDE [32], OCSVM [45], and HMMs) can greatly improves the overall performance. However, heterogeneous detectors use heterogeneous features for modeling and testing the normal behavior of a system. For example, OCSVM uses fixed-size vector based features while HMM and STIDE use fixed-size sliding window-based short sequences of system calls. Thus, profiling such features from large-scale traces of system calls is the very first and essential step before processing them by the ensemble of heterogeneous anomaly detectors.

For instance, each trace entry produced by the kernel collector [4], contains so many information related to each invoked system call such as arguments, result (return), process ID, process name, parent process ID, etc. Filtering and transforming such a large-scale trace of system calls into numerical sequences of system calls, and then, treating them to profile the heterogeneous features for heterogeneous anomaly detectors, is a time-consuming task for a single machine. To address this issue, a feasible solution would be to profile the heterogeneous features of the ensemble-based anomaly detection system by leveraging the power of existing parallel computation frameworks, such as HDFS (Hadoop Distributed File System) and the MapReduce programming model which are implemented on Big Data platforms.

[1] A crisp anomaly detector is the one that produces a decision (i.e., normal or anomalous) instead of scores (i.e., likelihood probability or similarity). This is contrasted with a soft detector, which produces scores instead of a decision. A soft detector can be converted into one or more crisp anomaly detectors by setting different thresholds on the output scores [3][4].

However, Hadoop with its original parallel computation model is technically not suitable for profiling sequential data due to dependencies on the temporal information or the orders of a sequence. For example, when HDFS splits a large trace file into two or more fixed-size blocks, Hadoop fails to keep track of the order or temporal information of large sequences within the trace file. To overcome this limitation, Li et. al, [24] have recently proposed an index pool data structure to predict time series by rolling a fixed-size window using Hadoop and the MapReduce programming model. Index pool has shown to be efficient in extracting the index key of a rolling window once the entire sequence is already distributed across multiple splits. However, extracting the index key for each rolling window gives rise to a linear increase of the computational time proportionally to the length of the sequence. Moreover, this approach can only profile the features of sliding windows, and thus, it is not suitable for the ensemble of heterogeneous anomaly detectors. Therefore, a more sophisticated MapReduce algorithm is required. This algorithm must profile the heterogeneous features such as fixed-size sliding windows for short sequences based anomaly detectors (e.g., HMMs and STIDE) and fixed-size feature vectors for the traditional machine learning based anomaly detectors (e.g., OCSVM).

In this paper, we propose an efficient anomaly detection approach called MASKED-A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System. MASKED has only one MapReduce job. It profiles the heterogeneous features from the large-scale traces of system calls, and then processes them by a pre-constructed set of Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER). In constructing BICKER, we use the same technique used in our previous work [28] with the exception of using the input of heterogeneous anomaly detectors (i.e., multiple HMMs, STIDE, and OCSVM) instead of homogeneous ones (i.e., only multiple HMMs). BICKER selects a set of diverse soft and their corresponding complementary crisp detectors which are used to construct the Boolean combination rules. Then, BICKER is used by MASKED to process the profiled heterogeneous features.

The main contributions of this paper are as follows:

- Construction of a set of Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER) by using the WPIBC Boolean combination technique [28]. BICKER takes heterogeneous anomaly detectors (i.e., multiple HMMs, STIDE, and OCSVM) as input instead of homogeneous ones (i.e., only multiple HMMs) as was the case in WPIBC.
- Selection of five most diverse soft anomaly detectors (i.e., three HMMs, STIDE, and OCSVM) where each one has six complementary crisp detectors, which are used to construct the final set of Boolean combination rules.
- A MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System (MASKED) that profiles heterogeneous features from large-scale traces of system calls and processes them using BICKER.

The rest of the paper is organized as follows. The next section surveys the state of the art in anomaly detection. Section 3 provides a discussion on different heterogeneous anomaly detectors and Big Data platforms (i.e., MapReduce & Hadoop). In Section 4, we describe the implementation of our proposed approach followed by the experimental results in Section 5. Finally, we conclude the paper in Section 6 and discuss the future directions.

II. Related Work

There exist several anomaly detection techniques and tools [8][36][37] in which the traces of system calls are used to detect the anomalous behavior at the host-level. The recent studies have also been showed that using the Big Data platforms, particularly, Hadoop and MapReduce programming model, improves the efficiency of system anomalies detection problem [16][24][26][34][35]. Among them, Matthews et al. [34] have recently proposed a MapReduce solution for detecting real-time anomalous behaviors in SCADA systems. They analyzed both the voltage and current phasors, as well as a set of frequency measurements to detect any deviations from the true value. However, this solution is technically not suitable for utilizing the power of MapReduce and Hadoop to profile short sub-sequences or time slice windows from a large-scale temporal data. This is due to the fact that the latter assume that the data should be preprocessed and stored in a CSV file before being used. Moreover, traditional machine learning approaches [45][47], use fixed-size feature vector instead of short sub-sequences. Therefore, this solution [34] is suitable for a single-based anomaly detector with a preprocessed time slice data and not appropriate for ensemble-based anomaly detection systems.

Zhenlong Li et al., [26] proposed a spatiotemporal indexing approach that can be used by a MapReduce job for retrieving and processing spatiotemporal climate data. They used the proposed index data structure as a global grid, which is accessed by each node for re-assembling the features from a block of data. However, the size of the global indexing grid increases exponentially with the increase of the spatiotemporal resolution (or time slice) size. Therefore, the spatiotemporal indexing is reliable when the time slice is large (e.g., daily basis). For a small window, however, the size of each global grid may reach several gigabytes which reduces the computational efficiency.

Kim et al., [16] proposed a host-based anomaly detection method by leveraging the Hadoop MapReduce parallel computation model in the era of host-generated Big Data. They reported that the behavior of malicious codes is logged basically on the host. They analyze the host log information which includes various log data such as enormous amounts of security logs, network and host information, and application transactions. This approach is also limited to profile only vector-based features. In that case, our proposed MapReduce solution, MASKED takes a full advantage of the parallel computation framework, Hadoop, by profiling heterogeneous features and processing them using a pre-constructed ensemble-based BICKER Boolean combination rules.

III. Background

This section provides background information on well-known system call based heterogeneous soft anomaly

detectors: STIDE, HMM, OCSVM, and ensemble-based Boolean combination techniques. The MapReduce paradigm as well as its implementation by Hadoop are also reviewed. The latter is used to implement the algorithm described in this work.

A. Heterogeneous Soft Anomaly Detectors

Most reported approaches for anomaly detection were based on sequence matching. During training, these approaches built the normal profile by segmenting the full-length sequences of system calls into fixed-length contiguous sub-sequences. They used a fixed-size sliding window which is shifted by one symbol at a time.

The very early approach was the Sequence Time Delay Embedding (STIDE) [33][38]. STIDE uses unique continuous sliding windows to construct the normal database which is a tree data structure. Moreover, it uses Hamming distance to measure similarity between two sub-sequences of system calls, and computes a score instead of a decision.

Hidden Markov Models (HMMs) have also been shown to provide a robust anomaly detection in sequences of system calls to the operating system kernel [7][10][28][42]. They are determined by the following three parameters in $\lambda = (A, B, \pi)$, where A represents the states and transition probability distribution, B represents the observation probability distribution of observation sequences that come from the temporal order of executions, and π represents the initial state probability distribution of each hidden state in a Markov process. Since the behavior of a process in UNIX or Windows system is represented as a discrete sequence of system calls, discrete HMM models are used [25] [29] to learn the behavior of a process. Typically, training an HMM using a discrete sequence of observations $\mathcal{O} = (\mathcal{O}_0, \mathcal{O}_1, \ldots, \mathcal{O}_{T-1})$ maximizes the likelihood function $P(\mathcal{O}|\lambda)$ over the parameter space represented by A, B, and π. The Baum-Welch (BW) algorithm [22] is the most commonly used Expectation-Maximization (EM) algorithm for estimating HMM parameters. It uses a Forward-Backward (FB) algorithm [29] at each iteration to efficiently evaluate the likelihood function $P(\mathcal{O}|\lambda,)$ and then updates the model parameters until the likelihood function stops improving or a maximum number of iterations is reached.

There are standard machine learning techniques such as the One-Class Support Vector Machine (OCSVM), which use fixed-size vectors as input features instead of sequence matching. The fixed-size vectors are generated from system call sequences using a technique known as bag of system calls [14][22][45][47]. The latter is adopted from text mining or information retrieval [15] where each unique system call (s_i) acts as a term or symbol of alphabet $\sum = \{s_1, s_2, \ldots s_m\}$ and the number of unique system calls is given by $m = |\sum|$ which is equal to the size of vectors. Let $T = \{s_1, s_2, \ldots s_L\}$ be a trace of a system call sequence of length L and $\mathcal{T} = \{T_1, T_2, \ldots T_K\}$ be a collection of K traces generated by an anomaly detection system. Each trace T_k is then encoded into a term vector $\boldsymbol{\mathcal{V}_k} < s_1, s_2, \ldots s_m >$ of size m, where each element (s_i) contains 1/0 based on the following condition as:

$$\boldsymbol{\mathcal{V}_k}(s_i) = \begin{cases} 1, & if\ s_i \in T_k \\ 0, & if\ s_i \notin T_k \end{cases}; \quad k = 1, \ldots K \tag{1}$$

The term vector $\boldsymbol{\mathcal{V}_k}$ can be weighted by the term frequency (*tf*) as follows:

$$\boldsymbol{\mathcal{V}_k}(s_i) = \Phi_{tf}(s_i, T_k) = freq(s_i) \tag{2}$$

where $freq$ is the number of occurrences of a system call s_i in T_k, normalized with the sequence length $L = |T_k|$. However, Φ_{tf} considers the discrimination ratio for each term only for a single sequence. To account for the discrimination ratio for each term over the whole K sequences, the term frequency inverse document frequency (*tf-idf*) can be used [14]. Moreover, the terms that are less frequent across the whole sequences are more uncertain, and thus, more informative. Therefore, the weighting measure *tf-idf* was used to compute the term vector $\boldsymbol{\mathcal{V}_k}$ as follows:

$$\boldsymbol{\mathcal{V}_k}(s_i) = \Phi_{tf-idf}(s_i, T_k, \mathcal{T}) = \frac{K}{df(s_i)} freq(s_i) \tag{3}$$

In traditional machine learning approaches such as the OCSVM, the sequential based system $\mathcal{T}(T_k, y_k)$ is first transformed into a fixed-size (m) vector-based system $\mathcal{X}(\boldsymbol{\mathcal{V}_k}, y_k)$ using equation (3). In this equation, y_i is the corresponding class labels (0/1) of each trace (T_k) of a system call sequence ($y_k = 0$, if T_K is "*normal*", otherwise 1, i.e., "*anomaly*"). The fixed-size vector-based system $\mathcal{X}(\boldsymbol{\mathcal{V}_i}, y_i)$ is then applied as input to the OCSVM model. The LIBSVM [2], a library for diverse types of SVM classifiers, is subsequently used to train the OCSVM model.

B. Ensemble-based Boolean Combination Techniques

The very first Boolean combination approach was proposed in [20]. The authors used only the AND (∧) and OR (∨) rules and fused all the responses in a ROC (Receiver Operating Characteristic) space [39]. The fused responses (i.e., the new emerging points on the ROC space) are then used to compute the composite ROC convex hull (ROCCH)[2]. On one hand, the combination of two diverse detectors (e.g., a best detector and a worst detector) using only these two rules may increase the false alarm rate [42]. On the other hand, the diversity among the combined two detectors is the main key factor for improving the detection accuracy [23].

The very first ensemble approach that considers the diversity is the Pair-wise Brute-force Boolean Combination (BBC2) by considering all the ten Boolean combination rules (a∧b, ¬a∧b, a∧¬b, ¬(a∧b), a∨b, ¬a∨b, a∨¬b, ¬(a∨b), a⊕b, a≡b). However, the pair-wise brute-force strategy is computationally intensive due to the high number of permutations. In this context, Wael et al. have proposed an

[2] All the points in a ROC space can be classified into two groups superior and inferior based on their *tpr* and *fpr* [5]. The ROC convex hull (ROCCH) is therefore the piece-wise outer envelope connecting only its superior points. The accuracy of a ROCCH curve is measured by the Area Under the Curve (AUC) [3].

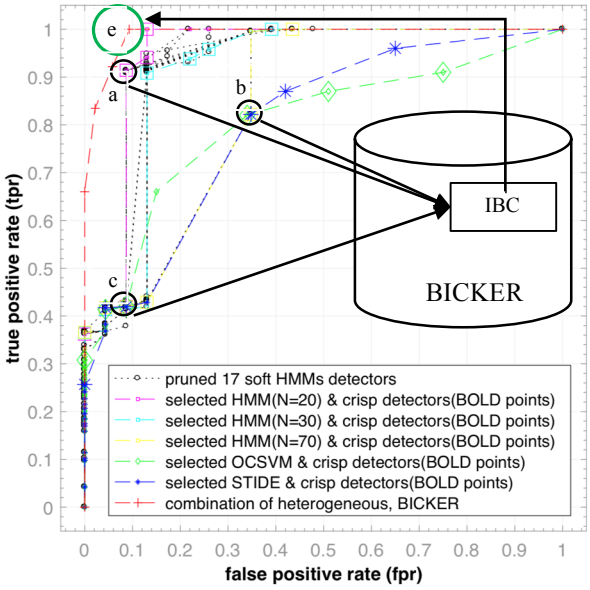


Fig. 1. The selected diverse heterogeneous soft anomaly detectors (OCSVM, STIDE, and 3 HMMs) including their corresponding complementary crisp detectors (bold marker points) and using one of the kappa-pruned ensemble based Weighted Pruning Iterative Boolean Combination (WPIBC) techniques [28].

Iterative Boolean Combination (IBC) method and obtained further improvement [27]. However, the sequence of combinations grows linearly with the increase of the number of iterations, which increases the complexity of the analysis [7]. Recently, we proposed a Weighted Pruning Iterative Boolean Combination (WPIBC) approach that first selects the most diverse detectors while pruning all the redundant ones before fusing the Boolean combination rules [28]. We also used WPIBC in constructing BICKER which is used by the proposed MapReduce solution.

C. MapReduce Programming Model and Hadoop

The MapReduce programming model uses split-apply-combine strategy for processing and generating Big Data with commodity hardware [21][41]. A MapReduce job is composed of two functions: Mapper and Reducer. The Mapper function reads each line of record from an input file, performs some operations, and produces a list of key-value pairs as output. The Reducer function takes all the intermediate values associated with a particular key, applies defined actions, and writes the results into the output files. Both Mapper and Reducer functions are designed to run simultaneously and independently on each node in a cluster.

Apache Hadoop [9] implements the MapReduce programming model with the distributed file system, known as Hadoop Distributed File System (HDFS). Hadoop splits a file into large blocks (typically, 64MB) and distributes them across several parallel nodes. Each node only accesses and processes the assigned data locally, which yields greater efficiency [17]. Moreover, Hadoop is scalable, fault tolerant, cost effective and flexible. As a result, it has become the industry standard for handling Big Data. A small Hadoop cluster has one master and multiple worker nodes. The master node contains JobTracker, TaskTracker, Name Node, and Data Node whereas the slave or worker node contains only TaskTracker and Data Node. The JobTracker initializes a MapReduce job and manages the TaskTracker on each node. The TaskTracker on each node executes the Mapper and Reducer tasks assigned by the JobTracker.

IV. Proposed Approach

In this work, we propose a MapReduce Solution for the Kappa-pruned Ensemble-based Anomaly Detection System (MASKED) that profiles the heterogeneous features from the large-scale traces of system calls, and then processes them by a pre-constructed set of Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER). In constructing BICKER, we leverage our previous proposed Weighted Pruning Iterative Boolean Combination (WPIBC) technique [28]. The only difference is that the inputs of BICKER are a set of heterogeneous soft anomaly detectors (e.g., multiple HMMs, STIDE, and OCSVM) whereas, WPIBC uses homogeneous ones (i.e., only multiple HMMs). BICKER is used by the proposed MapReduce solution (MASKED) to process the profiled heterogeneous features. MASKED is completely controlled by only one MapReduce job that not only profiles the heterogeneous features for the heterogeneous anomaly detectors (e.g., STIDE, HMM, and OCSVM) but also processes them by using BICKER Boolean combination rules. In the following, we first describe the construction procedure of BICKER and then, we present the proposed MapReduce solution.

A. Kappa-pruned Ensemble-based Iterative Boolean Combination Rules (BICKER)

Although, the construction procedure of BICKER is exactly the same as in WPIBC, the inputs of BICKER are now three main heterogeneous soft anomaly detectors (STIDE, multiple HMMs, and OCSVM) instead of only homogeneous multiple HMMs. We trained STIDE and HMM using the fixed-size sliding window based sequential features, and OCSVM using the *tf-idf* term vectors (both feature types can be profiled using the proposed MapReduce solution (MASKED) whose details are discussed in the next subsection B). We use the validation set same as in WPIBC [28] for selecting the most diverse soft and their corresponding complementary crisp detectors.

First, we compute a set of scores for each input soft anomaly detector. Then, we set all the possible thresholds on each set of scores. Each threshold is associated with a crisp detector that produces a set of responses 0/1 (0-means normal and 1-means anomaly), which in turn, produce a single point (*fpr*-false positive rate, *tpr*-true positive rate) on the ROC space. Therefore, each soft detector produces a set of crisp detectors or a set of points (*fpr, tpr*) on the ROC space with an AUC (area under the curve) value used as a performance metric for that soft detector.

With this setting and according to WPIBC [28], we select the most diverse soft detectors while pruning all the redundant ones using weighted kappa coefficients (an

extended version [12] of Cohen's kappa [19] that measures the degree of agreement between two soft detectors at the various ranks/levels/thresholds). Fig. 1 shows the selected five diverse base soft detectors (OCSVM, STIDE, and three HMMs) while pruning 17 redundant soft HMMs.

Let the number of possible thresholds be k. Each selected diverse base soft detector produces k crisp detectors. Then, we apply the MinMax-kappa pruning technique [7] on each selected soft detector. As a result, m (m<<k) complementary crisp detectors out of k candidate crisp detectors are selected while the trivial (always produces same responses either 0 or 1) and redundant crisp detectors are pruned. Fig. 1 illustrates the selected 6 complementary crisp detectors (bold marker points) from each selected diverse base soft detector.

The selected five diverse base heterogeneous soft anomaly detectors and their corresponding 30 complementary crisp detectors, are then used to construct the final Boolean combination rules. As in WPIBC, we leverage the IBC Boolean combination technique [42] in constructing BICKER. For instance, the ROC curve, red one with '+' marker points (shown in Fig. 1), is the resulting composite ROC curve using the BICKER Boolean combination rules on the validation set. In Fig. 1 and for simplicity, we show a composite emerging point (e) which results from the IBC combination of three selected complementary crisp detectors a, b, and c. The best-case scenario for BICKER would be to use only the five most diverse soft detectors or their corresponding 30 complementary crisp detectors to get this composite ROC curve. In contrast, when IBC is used without pruning, all the available 22 input soft detectors or 2,200 (in our case, k=100) crisp detectors should be used to get the same composite ROC curve [42].

Finally, we store the ensuing BICKER information into a NoSQL database: (i) the trained parameters of each selected soft detectors and the thresholds of their six complementary crisp detectors, and (ii) the constructed Boolean combination rules using only the selected complementary crisp detectors. The proposed MapReduce solution that contains only one MapReduce job, uses BICKER for processing the profiled heterogeneous features from a large-scale raw traces of system calls.

B. *Profiling Heterogeneous Features using Distributed File System*

It is well known that HDFS, a distributed file system, splits a large file (bigger than the block size, 64MB) into several fixed-size blocks, which are distributed across many parallel nodes [17]. However, if a trace file with a large sequence of system calls is stored into two or more HDFS

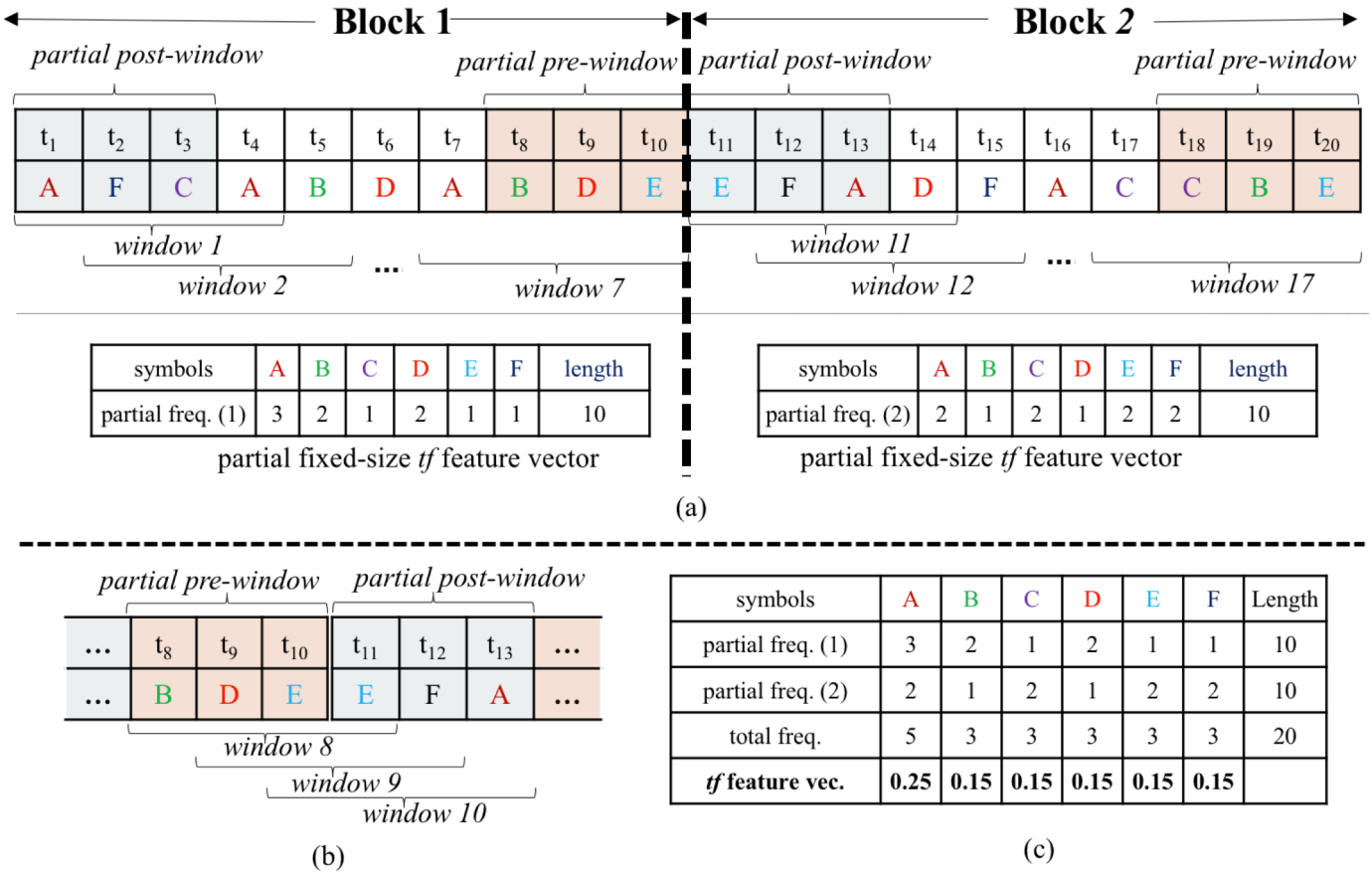


Fig. 2. A general approach for profiling heterogeneous features from a large-scale trace file that has a long sequence of system calls and stored in a distributed file system.

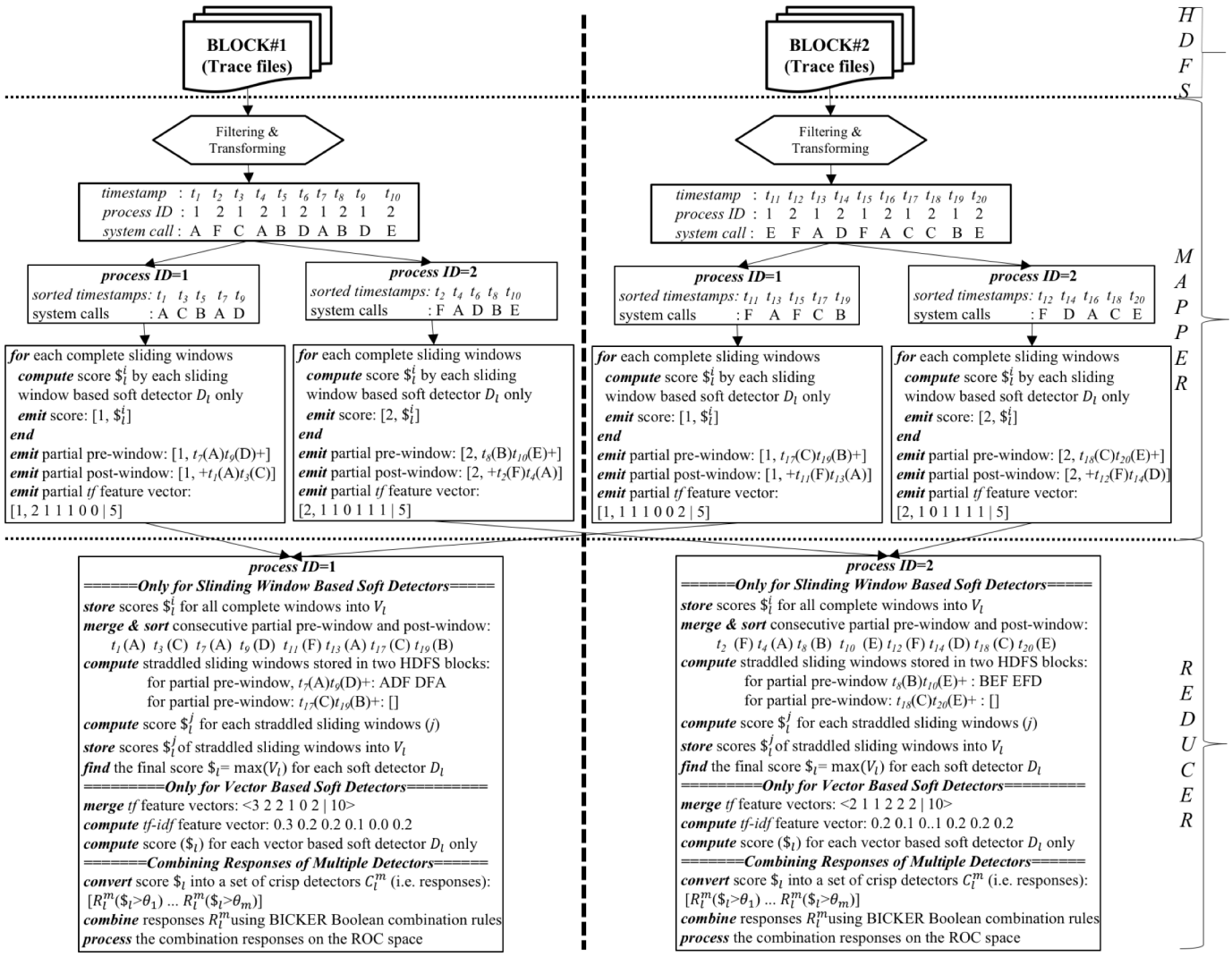


Fig. 3. The flow of data of the proposed MapReduce solution MASKED to profile the heterogeneous features of heterogeneous anomaly detectors and process them using a pre-constructed Kappa-pruned Ensemble based Iterative Boolean Combination Rules (BICKER).

blocks, the temporal orders of system calls will be lost. That is, some fixed-size sliding windows are straddled at the split boundary between two blocks [24]. Fig. 2 (a) shows an example in which three consecutive sliding windows (assuming a window of size four): window 8, window 9, and window 10 are straddled at the split boundary between two blocks. Indexing these straddle windows is important for re-assembling them at the aggregation level. In this work, we propose a general solution for indexing these straddle windows, which can be used for profiling both fixed-size sliding window based short subsequences as well as fixed-size feature vectors from a large-scale trace file that is stored in a distributed fashion.

Before profiling the fixed-size sliding windows, each distributed block produces a set of complete sliding windows including two partial windows (partial pre-window and partial post-window) as shown at the top of Fig. 2 (a). The main benefit of these two partial windows is that, at the aggregation level, only two consecutive partial pre-window and post-window are required to profile the rest of the straddle sliding windows. Fig. 2 (b) shows that the two-consecutive partial pre-window and post-window are merged into one partial subsequence before being sorted based on the timestamps (t). This partial subsequence is then used to produce the rest of the complete straddle sliding windows (windows 8, 9, and 10) at the split boundary between two blocks.

To profile the fixed-size feature vector, each block produces a partial *tf* feature vector whose size is fixed and equals the number of unique symbols used in the system. It also records the length of the processed subsequence (within a block) at the end of that partial *tf* vector. The bottom of Fig 2 (a) shows two blocks producing two partial *tf* feature vectors with size of six (i.e., the number of unique symbols: A B C D E F), excluding the last element which is the length (10) of the processed subsequence. At the aggregation level,

Fig. 2 (c) shows that the two partial *tf* feature vectors are also merged into a complete *tf* feature vector, normalized by the total length (20) of that sequence. The *tf* feature vector is then transformed into *tf-idf* feature vector using equation (3) and the precomputed document frequency (*df*).

C. A MapReduce Solution for Profiling and Processing Large-scale Traces of System Calls

In the proposed method, we use a set of system call traces collected by the Anubis emulator tool and stored in HDFS, as a large-scale dataset [4][13]. Running under Windows operating system, the OS emulator has a kernel module that tracks system call events and annotates them according to privacy rules. Fig. 3 shows the flow of data of the MapReduce job that extracts, transforms, and profiles the heterogeneous features, and then, processes them using BICKER (as discussed in subsection A).

Since each trace file contains so many information related to each invoked system call (e.g., result, pid, process name, and parent process ID), the mapper function first filters and transforms a raw system call trace file into a set of tuples. Each tuple contains three fields: ⟨timestamp, pid, system_call⟩ which are needed to profile the heterogeneous features for the anomaly detectors. As shown in Fig, 3, the mapper function groups all the tuples into sub-sequences of system calls based on each process ID. The Mapper function then computes all the complete sliding windows, including the two partial windows. It also computes a partial *tf* feature vector for each sub-sequence of system calls.

Once a sliding window, w_i is complete, the mapper function accesses BICKER to load the trained parameters of each sliding window based soft detectors, D_l (e.g., STIDE and three HMMs). Then, it uses them to compute the score $\$_l^i$. The score is then sent as a *key*-*value* pair into the reduce function, where *key* is the pid and *value* is the score. If the sliding window is partial, the score is not computed, and the partial window is sent as a value together with the pid to the reduce function. Similarly, the mapper function directly sends a partial *tf* feature vector as a (*key*, *value*) pair into the reduce function, where *key* is the pid and *value* is the partial *tf* feature vector.

For each process, the reduce function re-assembles (i.e., merges and sorts) the partial windows and uses them to compute the straddled sliding windows which were stored in two HDFS blocks. It also computes the scores ($\$_l^j$) for each straddled sliding windows (w_j) by accessing each sliding window-based soft detectors (D_l) from BICKER. Then, it aggregates all the scores V_l=[$\$_l^i$ $\$_l^j$] to find the maximum which is considered as the desired score $\$_l$=max($V_l$), for each sliding window based soft detectors (D_l). For each process, the reduce function aggregates all the partial *tf* vectors into a single *tf* vector, normalized with the length of the sequence. The normalized *tf* vector is further weighted by the document frequency (df) and transformed into *tf-idf* feature vector using equation (3). The *tf-idf* feature vector is then processed by accessing each vector based soft detectors, D_l (e.g., OCSVM) from BICKER to compute the score $\$_l$.

The reduce function accesses BICKER to load the thresholds of each soft detector, D_l, and converts the computed score $\$_l$ into a set of six (m=6) complementary crisp detectors C_l^m. Then, the responses R_l^m of each crisp detector C_l^m are combined using the BICKER Boolean rules. Finally, the combination responses R_l^m are used to compute the final composite ROC convex hull (ROCCH) on the ROC space.

V. Experiments and Results

To access the performance of the proposed MapReduce solution, a small cluster with only seven nodes was used as a platform. The CANALI-WD [13] was used as raw traces of system calls dataset.

A. CANALI-WD Dataset

CANALI-WD dataset consists of two normal datasets called Goodware and Anubis-good and two malware datasets called malware and malware-test. The Goodware dataset contains a massive amount of 180 GB execution traces collected from 10 different machines. The Anubis-good dataset contains the traces of 36 benign applications executed under Anubis [1]. The malware dataset is a collection of 6,000 malware execution traces including a mix of all the existing categories (botnets, worms, dropper, Trojan horses, etc.). In addition, it is composed of 1,200 malware execution traces collected from a different machine, excluding the normal ones used for Anubis.

B. Setting the Training Parameters

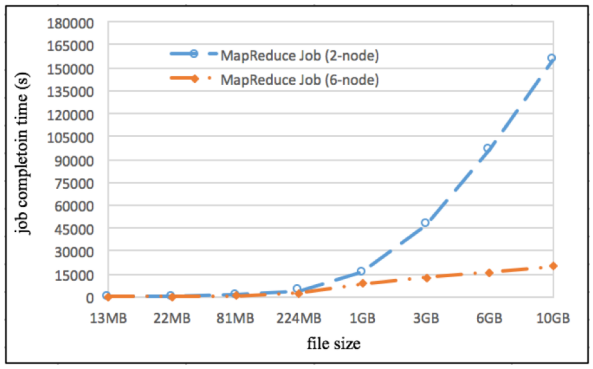
For training, we used the traces of normal behavior of Anubis-good and Goodware datasets (excluding the traces of machine 10, which are used for testing). In addition to the traces of machine 10, malware and malware-test datasets were used to construct the testing set with varied sizes to evaluate the performance of MASKED. Among the evaluation traces, we randomly selected 10% from machine 10, malware, and malware-test datasets to form the validation set. The training dataset was used to train the three-main heterogeneous soft anomaly detectors (STIDE, multiple HMMs, and OCSVM). In the case of STIDE, we built the normal database using the normal unique short-sequences. We also used the same unique normal short-sequences to train the HMM parameters (A, B, π) using the BW algorithm [22]. In the case of OCSVM, we converted the normal training sequences into the *tf-idf* feature vectors using Equation (3). The converted *tf-idf* vectors were used to train the OCSVM using the Gaussian or RBS (radial basis function) kernel function [2]. We obtained the best accuracy for both OCSVM and OCSVM on the validation set for sigma = 0.001.

To select the best window size for both STIDE and HMM, we trained them with three different window sizes (5, 10, and 20). We obtained the best accuracy using the validation set for a window size of 5 which was selected as the default window size. Moreover, to find the well-trained HMMs

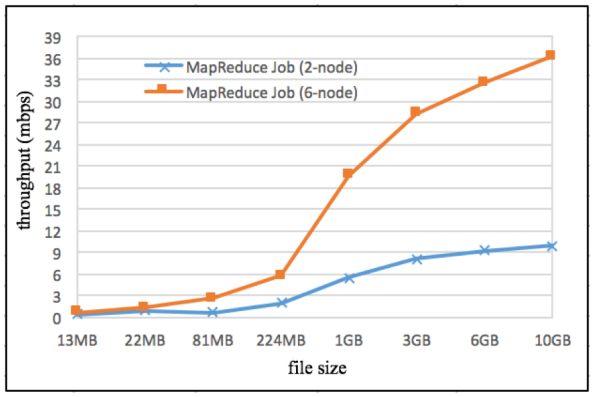
models, we trained different discrete-time ergotic HMMs with various *N* values (N=10, 20, … 100) [28].

C. Cluster Configuration

We configured a small Hadoop cluster with only seven nodes to test the proposed approach. We used Matlab Distributed Computing Server [3] to setup this small cluster. Among the seven nodes, six nodes were used as a Hadoop



(a)



(b)

Fig. 4. Performance comparison between 6-node and 2-node Hadoop clusters: (i) job completion time and (ii) throughput.

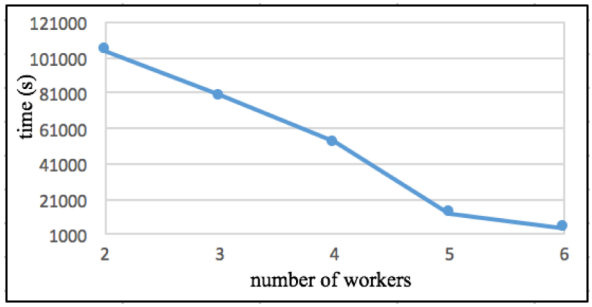


Fig. 5. Performance comparison for different numbers of workers, and a file size of 10 GB.

cluster and one node was used as a database server to store the contents of BICKER. The five nodes of Hadoop cluster (excluding the Hadoop master node) were used to accumulate a large-scale system call traces dataset. The Hadoop cluster

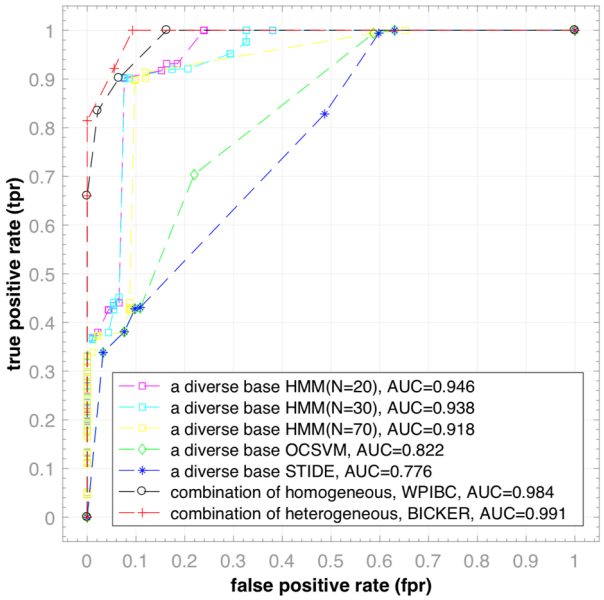


Fig. 6. Comparison of the combination results on the ROC space using the standard AUC (Area Under the Curve) as a measurement metric.

with six nodes was used as a HDFS with block sizes of 64MB each.

D. Analyzing Performance of the Proposed MapReduce Solution

We evaluated the performance of the proposed MapReduce solution by varying the input file size from 13MB to 10GB. We compared the performance of 6-node and 2-node Hadoop cluster settings in terms of job completion time (seconds) and throughput (MBps). Fig. 4 shows the performance of the MapReduce job with different file sizes. According to Fig. 4 (a), when the file size is very small (up to 81MB), the completion times are almost constant. When the input file size increases above 81MB, however, our approach gives rise to a significant reduction of the completion time with a 6-node cluster compared to 2-node cluster. For example, when the file size is 10GB, the completion times were 20,068s and 155,187s for 6-node and 2-node cluster settings, respectively. That is, MapReduce job with 6-node cluster is approximately 8 times faster than that with 2-node cluster.

In terms of throughput and according to Fig. 4(b), we can see that when the file size is more than 224MB, the 6-node cluster far outperformed the 2-node cluster. For example, when the file size of 10GB, the 6-node cluster achieved a throughput of 36MBps compared to 9MBps achieved by the 2-node cluster. That is, the throughput of the 6-node cluster is about 4 times higher than that of the 2-node cluster.

We evaluated the scalability of MASKED when the number of worker nodes is increased. According to Fig. 5, we can see that the MapReduce job reduces the completion time inversely proportional to the number of worker nodes. This result was expected for two reasons: 1) Hadoop is known for its scalability; and 2) MapReduce parallel/distributed

computing provides a powerful solution for accessing, processing, grouping, and aggregating a large-scale data such as the one used in this study.

We analyzed the outputs (i.e., combination responses) of the proposed MapReduce job on the ROC space. Fig. 6 shows the achieved composite ROCCH (red color) after combining the responses of the selected complementary crisp detectors. According to this figure, BICKER shows a significant improvement compared to the performance of the individual detectors, particularly, when the false alarm is close to zero. These results show conclusively that using heterogeneous detectors gives rise to better anomaly detection accuracy than that with homogeneous multiple HMMs.

E. *Effects of Partial Pre(Post)-window for Indexing the Straddle Sliding Windows*

Instead of using an additional indexing data structure like in [24][26], we used two partial windows which are essential for indexing the straddle sliding windows at the split boundary between two HDFS blocks. In contrast, Li method [24] needs to access an index pool data structure to profile each complete and partial window.

At the aggregation end, the reduce function only accesses the two consecutive partial windows to produce the remaining complete windows at the split boundary between two blocks (Fig. 2). This is in contrast with Li method in which both the mapper and reducer need to maintain many partial windows to produce the remaining complete windows. In addition, the proposed approach does not need to store and access any additional index pool data structure as the case in [24][26].

F. *Effects of Heterogeneous Detectors in Constructing the Boolean Combination Rules, BICKER*

It is well-known that the diversity between two combined crisp or soft detectors is an important factor for any ensemble-based anomaly detection approach [7][23][28][44]. That is, if the responses of two crisp detectors are comparable, combining them using Boolean combination rules declines the anomaly detection accuracy. In our previous work [28], we developed a Boolean combination approach (WPIBC) which demonstrated how the diversities among the combined soft and crisp detectors can be guaranteed. This work shows that the diversity is improved when using heterogeneous detectors.

VI. Conclusion and Future Work

We proposed an efficient MapReduce solution, namely called MASKED, for the Kappa-pruned Ensemble-based Anomaly Detection Systems. MASKED has only one MapReduce job that profiles the heterogeneous features from large-scale raw traces of system calls for heterogeneous anomaly detectors. The MapReduce job also processes the profiled heterogeneous features using a constructed kappa-pruned iterative Boolean combination rules (BICKER). The experimental results with varied sizes of HADOOP clusters, have shown that MASKED is efficient and scalable for detecting system anomaly with the help of kappa-pruned ensemble-based anomaly detection system. In the future, we plan to evaluate the efficiency of MASKED with more worker nodes and anomaly detectors.

To the best of our knowledge, MASKED is the first initiative where MapReduce is used to profile and process the heterogeneous features for heterogeneous anomaly detectors.

References

[1] [Online]. Available: http://anubis.iseclab.org [Accessed: 15 June 2017].

[2] [Online]. Available at: https://www.csie.ntu.edu.tw/~cjlin/libsvm/ [Accessed: 10 June 2017].

[3] [Online]: https://www.mathworks.com/help/mdce/configure-a-hadoop-cluster.html [Accessed: 5 June 2017].

[4] A. Lanzi, M. Christodorescu, D. Balzarotti, E. Kirda, and C. Kruegel, "AccessMiner: Using System-Centric Models for Malware Protection," in Proc. of the 17th ACM Conf. on Computer and Communications Security, pp. 399-412, 2010.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," Journal of the Royal Statistical Society, Series B (Methodological), vol. 39, no. 1, pp 1-38, 1977.

[6] A. Pramod, T. Krishnaprasad, M. Surendra, S. Amit, and B. Tanvi. "Understanding City Traffic Dynamics Utilizing Sensor and Textual Observations," in Proc. of the 13th AAAI Conf. on Artificial Intelligence, pp. 3793-3799, 2016.

[7] A. Soudi, W. Khreich, and A. Hamou-Lhadj, "An Anomaly Detection System based on Ensemble of Detectors with Effective Pruning Techniques," IEEE Int. Conf. on Software Quality, Reliability and Security, pp.109-118, Aug. 2015.

[8] A. Sultana, A. Hamou-Lhadj, S. Murtaza, and M. Couture, "An Improved Hidden Markov Model for Anomaly Detection Using Frequent Common Patterns," in Proc. of the IEEE Int. Conf. on The Communication and Information Systems Security Symposium, pp. 1113-1117, 2012.

[9] Apache Hadoop. [Online]. Available: http://hadoop.apache.org/ [Accessed: 20 June 2017].

[10] B. Gao, H. Y. Ma, and Y.H. Yang, "HMMs (Hidden Markov Models) based on Anomaly Intrusion Detection Method," in Proc. of 2002 Int. Conf. on Machine Learning and Cybernetics, vol. 1, pp. 381-385, 2002.

[11] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in Proc. of the IEEE Computer Society Symposium on Research in Security and Privacy, (Oakland, CA, USA), pp. 133-45, 1999.

[12] C.A.M. Valiquette, A.D. Lesage, and C. Mireille, "Computing Cohen's Kappa coefficients using SPSS MATRIX," Behavior Research Methods, Instruments, & Computers, vol. 26, no. 1, pp. 60-61, 1994.

[13] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," in Proc. of the 2012 Int. Symp. on Software Testing and Analysis (ISSTA 2012), ACM, New York, NY, USA, pp. 122-132, 2012.

[14] D.K. Kang, D. Fuller, and V. Honavar, "Learning Classifiers for Misuse Detection Using a Bag of System Calls Representation," Lecture Notes in Computer Science, vol 3495, pp. 511-516, 2005.

[15] G. Salton, "Automatic text processing," Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[16] H. Kim, J. Kim, and I. Kim, "Behavior-based anomaly detection on big data," in Proc. of the 13th Australian Information Security Management Conf., pp. 73-80, Dec. 2015.

[17] IBM, "What is the Hadoop Distributed File System (HDFS)?," [Online]. Available: www.ibm.com/software/data/infosphere/Hadoop/ [Accessed: 25 June 2017].

[18] J. Cohen, "A coefficient of agreement for nominal scales," Educational & Psychological Measurement, vol. 20, pp. 37-46, 1960.

[19] J. Cohen, "A coefficient of agreement for nominal scales," Educational & Psychological Measurement, vol. 20, no. 1, pp. 37-46, 1960.

[20] J. Daugman, "Biometric decision landscapes," Cambridge U., UK, Tech. Rep. UCAM-CL-TR-482, 2000.

[21] J. Dean, and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107-113, 2008.
[22] L. E. Baum, G. S. Petrie, and N. Weiss, “A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains,” The Annals of Mathematical Statistics, vol. 41, no. 1, pp. 164-171, 1970.
[23] L. I. Kuncheva, “A bound on kappa-error diagrams for analysis of classifier ensembles,” IEEE Trans. on Knowledge and Data Engineering, vol. 25, no. 3, pp. 494-501, 2013.
[24] L. Li, F. Noorian, D.J.M. Moss, and P.H.W. Leong, “Rolling window time series prediction using MapReduce,” in Proc. of the 2014 IEEE 15th Int. Conf. on Information Reuse and Integration (IEEE IRI 2014), pp. 757-764, 2014.
[25] L. Rabiner, “A tutorial on Hidden Markov Models and selected applications in speech recognition,” in Proc. of the IEEE, vol. 77, no. 2, pp. 257-286, 1989.
[26] L. Zhenlong, H. Fei, J. L. Schnase, D. Q. Duffy, L. Tsengdar, M. K. Bowen, and Y. Chaowei, “A spatiotemporal indexing approach for efficient processing of big array-based climate data with MapReduce,” Int. Journal of Geographical Information Science, vol. 31, no.1, pp. 17-35, 2017.
[27] M. Barreno, A. Cardenas, and D. Tygar, “Optimal roc for a combination of classifiers. In Advances in Neural Information Processing Systems (NIPS),” MIT Press, pp. 57-64, Jan. 2008.
[28] M. S. Islam, W. Khreich, and A. Hamou-Lhadj, “Anomaly Detection Techniques Based on Kappa-Pruned Ensembles,” in IEEE Trans. on Reliability, vol. 67, no. 1, pp. 212-229, March 2018.
[29] M. Stamp, “A Revealing Introduction to Hidden Markov Models,” Dec. 2015.
[30] P. Wang, L. Shi, B. Wang, Y. Wu, and Y. Liu, “Survey on HMM based anomaly intrusion detection using system calls,” in 5th Int. Conf. on Computer Science and Education (ICCSE), pp. 102-105, 2010.
[31] S. Aljawarneh, M. Aldwairi, and M.B. Yassein, “Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model,” Journal of Computational Science, 2017.
[32] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, “A sense of self for Unix processes,” in Proc. of the 1996 IEEE Symp. on Research in Security and Privacy, pp. 120-128, 1996.
[33] S. Forrest, S. Hofmeyr, and A. Somayaji, “The evolution of system call monitoring,” in Computer Security Applications Conference (ACSAC), pp. 418-430, Dec. 2008.
[34] S. Matthews, and A. S. Leger, “Leveraging MapReduce and Synchrophasors for real-Time anomaly detection in the smart grid,” in IEEE Trans. on Emerging Topics in Computing, pp. 1-1, 2017.
[35] S. S. Murtaza, A. Hamou-Lhadj, W. Khreich, and M. Couture, “TotalADS: Automated Software Anomaly Detection System,” in Proc. of the 14th IEEE Int. Conf. on Source Code Analysis and Manipulation (SCAM), 2014.
[36] S. S. Murtaza, A. Sultana, A. Hamou-Lhadj, and M. Couture, “On the Comparison of User Space and Kernel Space Traces in Identification of Software Anomalies,” in Proc. of the 16th European Conf. on Software Maintenance and Reengineering (CSMR'12), pp.127-136, 2012.
[37] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture, “A Host-based Anomaly Detection Approach by Representing System Calls as States of Kernel Modules,” in Proc. of the 24th IEEE Int. Symp. on Software Reliability Engineering (ISSRE), pp. 431-440, 2013.
[38] S.A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion detection using sequences of system calls,” Journal of Computer Security," vol. 6, no. 3, pp. 151-180, 1998.
[39] T. Fawcett, “An introduction to ROC analysis,” Pattern Recognition Letter, vol. 27, no. 8, pp. 861-874, 2006.
[40] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection: A Survey,” ACM Computing Surveys, vol. 41, no. 3, pp. 1-58, 2009.
[41] W. Hadley, “The split-apply-combine strategy for data analysis,” Journal of Statistical Software, vol. 40, no. 1, pp. 1-29, 2011.
[42] W. Khreich, E. Granger, A. Miri, and R. Sabourin, “Boolean combination of classifiers in the ROC space,” in 20th Int. Conf. on Pattern Recognition, Istanbul, Turkey, pp. 4299-4303, Aug. 2010.
[43] W. Khreich, E. Granger, R. Sabourin, and A. Miri, “Combining Hidden Markov Models for anomaly detection,” Int. Conf. on Communications (ICC), Dresden, Germany, pp. 1-6, June 2009.
[44] W. Khreich, S.S. Murtaza, A. Hamou-Lhadj, and C. Talhi, “Combining heterogeneous anomaly detectors for improved software security,” Journal of Systems and Software, vol. 137, pp. 415-429, March 2018.
[45] W.H. Chen, S.H. Hsu, H.P. Shen, “Application of SVM and ANN for intrusion detection,” Computers & Operations Research, vol. 32, no. 10, pp. 2617-2634, 2004.
[46] X. Hoang and J. Hu, “An efficient hidden Markov model training scheme for anomaly intrusion detection of server applications based on system calls,” in Proc. of the IEEE Int. Conf. on Networks (ICON), Singapore, vol. 2, pp. 470-474, 2004.
[47] Y. Liao, and V.R. Vemuri, “Use of K-Nearest Neighbour classifier for intrusion detection,” Computers & Security, vol. 21, no. 5, pp. 439-448, 2002.