Combining Heterogeneous Anomaly Detectors for Improved Software Security

Wael Khreich\textsuperscript{a,\star}, Syed Shariyar Murtaza\textsuperscript{a}, Abdelwahab Hamou-Lhadj\textsuperscript{a}, Chamseddine Talhi\textsuperscript{b}

\textsuperscript{a}Software Behaviour Analysis (SBA) Research Lab, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada
\textsuperscript{b}Department of Software Engineering and Information Technology, École de technologie supérieure, Montreal, Canada

Abstract

Host-based Anomaly Detection Systems (ADSs) monitor for significant deviations from normal software behavior. Several techniques have been investigated for detecting anomalies in system call sequences. Among these, Sequence Time-Delay Embedding (STIDE), Hidden Markov Model (HMM), and One-Class Support Vector Machine (OCSVM) have shown a high level of anomaly detection accuracy. Although ADSs can detect novel attacks, they generate a large number of false alarms due to the difficulty in obtaining complete descriptions of normal software behavior. This paper presents a multiple-detector ADS that efficiently combines the decisions from heterogeneous detectors (e.g., STIDE, HMM, and OCSVM), using Boolean combination in the Receiver Operating Characteristics (ROC) space, to reduce the false alarms. Results on two modern and large system call datasets generated from Linux and Windows operating systems show that the proposed ADS consistently outperforms an ADS based on a single best detector and on an ensemble of homogeneous detectors. At an operating point of zero percent false alarm rate, the proposed multiple-detector ADS increased the true positive rate by 500\% on the Linux dataset and by 25\% on the Windows dataset. Furthermore, the combinations of decisions from multiple heterogeneous detectors make the ADS more reliable and resilient against evasion and adversarial attacks.

Keywords: Anomaly Detection Systems, Intrusion Detection Systems, Heterogeneous and Reliable Systems, Decision-Level Combination.

\textsuperscript{\star}Corresponding author
\textit{Email addresses:} wkhreich@ece.concordia.ca (Wael Khreich), smurtaza@ece.concordia.ca (Syed Shariyar Murtaza), wahab.hamou-lhadj@concordia.ca (Abdelwahab Hamou-Lhadj), chamseddine.talhi@etsmtl.ca (Chamseddine Talhi)

1. Introduction

Intrusion Detection Systems (IDSs) are used to identify and report unauthorized or suspicious computer or network activities. Host-based IDSs are designed to monitor the host system activities and states, while network-based IDSs monitor network traffic for multiple hosts. In either case, an IDS can be classified into misuse detection or anomaly detection depending on whether the intrusion features are known or not during the design phase. Unlike misuse detection techniques, which look for patterns of known attacks, anomaly detection is capable of detecting novel attacks. An anomaly detection system (ADS) constructs a profile of expected normal behavior using datasets that are collected over a period of normal (attack-free) activity. During operation, the ADS attempts to detect events that deviate significantly from the expected normal profile. These deviations are considered and reported as anomalous events; however, they are not necessarily malicious activities because they may also correspond to coding or configuration errors.

Host-based ADSs, the focus of this paper, typically monitor for significant deviations in operating system calls, as they provide a gateway between user and kernel mode [1, 2]. The temporal order of system calls issued by a process to request kernel services, have been shown effective in describing normal process behavior [1, 2]. A substantial amount of research has focused various statistical, neural, machine learning and data mining techniques for detecting anomalies in system call sequences (see survey in [3]). Among these, sequential learning techniques such as the Sequence Time-Delay Embedding (STIDE) [1, 2, 3] and Hidden Markov Models (HMMs) [4, 5, 6, 7, 2, 8, 9, 10, 11, 12, 13, 14] have been shown to successfully model the sequential nature of system calls, and hence provide a high level of anomaly detection accuracy.

Standard machine learning techniques, such as Support Vector Machines (SVM) have also been proposed for detecting system call anomalies. These techniques require a fixed-size feature vector as an input for training. The bag of system calls is the most commonly used mapping to transform a trace of system calls into a feature vector of binary flags [15, 16, 17].

Despite over two decades of research effort, ADSs based on system calls still face several challenges that limit their deployment in commercial settings. In practice, ADSs generate an excessive number of false alarms, which undermine their credibility. False alarms are due in large part to the difficulty in obtaining complete descriptions of normal system behavior and to the changes that may occur in the monitored environment over time. Combining the predictions from multiple diverse and accurate detectors reduces the risk of selecting a single detector with poor generalization performance. Different detectors may commit different errors and hence provide complementary information that can be exploited by the combiner to reduce the number of false alarms [18, 19].

In previous work, we proposed an anomaly detection technique that combines the scores of several homogeneous detectors in the Receiver Operating Characteristics (ROC) space and explored its potential in improving the accuracy of an ADS [20]. More precisely, the approach combines multiple HMMs
each with a varying number of hidden states. Although the approach showed promising results, it suffered from the following drawbacks. First, it has been shown in the literature on machine learning (see [21]) that varying the parameters of homogeneous detectors provides a less diverse ensemble for combination, and hence a reduced overall detection accuracy, than that provided by heterogeneous detectors. Although the combination of several HMMs makes the ADS more resilient to evasion attacks than a single-detector system, the combination of multiple heterogeneous detectors further improves the ADS resilience to evasion and adversarial attacks (as detailed in Section 5). Furthermore, the previous approach was validated using the University of New Mexico (UNM) system call datasets [2], which are 20 years old. They were still in use due to the lack of publicly available system call benchmarks at that time. In addition, training HMMs with large number of states on large datasets requires a considerable amount of time and resources. The time complexity for training an HMM scales linearly with the sequence length and quadratically with the number of states, and its memory complexity scales linearly with both sequence length and number of states.

To address these issues, we propose, in this paper, a new anomaly detection system that combines powerful heterogeneous detectors, namely STIDE, HMM and One-Class SVM (OCSVM) detectors. We show that the combination of responses from heterogeneous detectors can reduce the number of false alarms while increasing the detection accuracy (without a significant performance overhead) compared to the use of homogeneous detectors. To support our findings, we conducted experiments using system call traces from two modern and large datasets generated from Linux and Windows operating systems. The first is a recently proposed system call dataset, called ADFA Linux Dataset (ADFA-LD) [22]. It has been made publicly available on the website of the University of New South Wales (UNSW) [22]. The ADFA-LD system call traces are generated from modern Linux operating systems, using contemporary hacking techniques. Furthermore, another dataset containing a large collection of system call traces generated from several Windows machines is provided by Canali et al. [23]. We refer to this dataset as CANALI-WD. These datasets provide a more realistic benchmark for evaluation of Host-based ADSs.

The experimental results (in Section 4, Figures 3 and 4) show that, at an operating point of zero percent false alarm rate, our heterogeneous multiple-detector ADS has significantly increased the true positive rate by 500% on the ADFA-LD and by 25% on the CANALI-WD datasets, compared to existing approaches including the one we proposed in [20]. Moreover, the paper provides an in-depth discussion on the advantages of combining the decisions from multiple heterogeneous detectors, in particular, making the ADS more resilient and robust to evasion and adversarial attacks, a topic that has not received much attention in the literature. Finally, a detailed analysis of time and memory complexity is provided for both design and operational phases.

The rest of this paper is organized as follows. The next section reviews the application of STIDE, HMM, and OCSVM detectors in system call based ADSs. In Section 3, the iterative Boolean combination technique is presented
and an illustrative example is provided for further clarification. The experimental methodology in Section 4 describes the datasets and the evaluation metrics. The simulation results are presented and discussed in Section 4, followed by a discussion on adversarial attacks and evasion techniques in Section 5. Finally, the conclusions and future works are presented in Section 6.

2. Anomaly Detection using System Call Sequences

Forrest et al. were the first to suggest that the temporal order of system calls could be used to represent the normal behavior of a privileged process [1]. They have collected system call datasets from various privileged processes at the University of New Mexico (UNM), and confirmed that short sequences of system calls are consistent with normal process operation, and unusual burst will occur during an attack.

The authors proposed a host-based ADS using a sequence matching techniques, called STIDE (Sequence Time-Delay Embedding), for detecting anomalous system call sequences generated by UNIX privileged processes [1]. To build a profile of normal behavior for a process of interest, STIDE proceeds by segmenting and enumerating the system call traces provided for training into fixed-length contiguous sequences, using a fixed-size sliding window, shifted by one symbol. These sequences are then stored in a database that represents the normal process behavior. During operations, STIDE uses the sliding window scheme (with the same window size used to construct the normal profile) to scan the system calls generated by the monitored process for anomalies – sequences that are not found in the normal database.

STIDE is therefore a crisp (or binary) detector that declares any sequence not found in the normal database as anomalous. Therefore rare normal sequences that did not appear in the normal database will be misclassified as anomalies (producing large numbers of false alarms). To reduce the number of false alarms, an anomalous score is defined as the number of mismatches in a temporally local region, and then an arbitrary threshold is set on this score, above which a sequence is considered as anomalous [2]. Hamming distance between sequences – the number of positions in which two sequences differ – has been successfully employed as an alternative measure of anomalous behavior [24]. To compute the anomaly score for a new observation sequence \( o_1, o_2, \ldots, o_T \) (consisely denoted by \( o_{1:T} \)), the Hamming distance between between \( o_{1:T} \) and all sequences in the normal database is first computed; then the minimal Hamming represents the anomalous score. STIDE is now transformed to a soft detector, which outputs scores instead of binary decisions as further described in Section 3. While the time complexity for STIDE to process a trace of length \( T \) using a window of size \( W \) is of the order of \( O(W.T) \), its worst-case memory complexity is \( O(S_u.W) \), where \( S_u \) is number of unique sequences extracted from \( T \). However, in practice, the storage requirements are much lower because the sequences are stored as trees [24].

Several statistical and machine learning techniques have been investigated over the last two decades for detecting system call anomalies using the UNM
data sets [3]. For instance, finite state automata have been proposed to model the system calls language, using deterministic or nondeterministic
automata [25, 26], or a call graph representation [27]. Lee et al., evaluated information-theoretic measures such as entropy and information cost [28]. Abstracting the system call traces as interactions of kernel state modules has also been proposed to reduce the processing time [29, 30]. Application of machine learning techniques include neural network [31], k-nearest neighbors [32], Markov models or n-grams [33, 34], Bayesian models [35]. Among these, techniques based on discrete HMMs have been shown to produce a high level of detection accuracy on the UNM system call datasets [2, 8, 9, 10, 13, 14, 11, 12, 6, 7, 36].

A discrete HMM is a stochastic process for sequential data [5, 37]. An HMM is determined by two interrelated mechanisms— a latent Markov chain having a finite number of states \( N \), and a set of observation probability distributions, each one associated with a state. Starting from an initial state \( S_i \in \{S_1, ..., S_N\} \), determined by the initial state probability distribution \( \pi_i \), at each discrete-time instant, the process transits from state \( S_i \) to state \( S_j \) according to the transition probability distribution \( a_{ij} \). The process then emits a symbol \( v_k \), from a finite alphabet \( V = \{v_1, ..., v_M\} \) of size \( M \) symbols, according to the output probability distribution \( b_j(v_k) \) of the current state \( S_j \). HMM is commonly parametrized by \( \lambda = (\pi, A, B) \), where the vector \( \pi = \{\pi_i\} \) is the initial state probability distribution, matrix \( A = \{a_{ij}\} \) is the state transition probabilities, and matrix \( B = \{b_j(v_k)\} \) is the output probabilities, \((1 \leq i, j \leq N \text{ and } 1 \leq k \leq M)\).

A well trained HMM provides a compact detector that captures the underlying structure of a process based on the temporal order of system calls, and detects deviations from normal system call sequences with high accuracy and tolerance to noise. Training an HMM from a sequence (or a block) of observation symbols, \( o_{1:T} \), aims at estimating HMM parameters \( \lambda \) to best fit the training data. Typically, parameters estimation consists of maximizing the likelihood of the training data over HMM parameters space, \( P(o_{1:T} | \lambda) \). Since this likelihood depends on the latent states, there is no known analytical solution to the learning problem. Iterative optimization techniques, such as the Baum-Welch (BW) algorithm, are applied to estimate the HMM parameters over several training iterations, until the likelihood function is maximized [5, 37]. During operation, the likelihood of a new observation sequence \( o_{1:T} \) given a trained HMM \( \lambda \), \( P(o_{1:T} | \lambda) \) is typically evaluated by using the Forward-Backward (FB) algorithm [5, 37]. Finally, a third canonical property of HMM is called decoding or finding the most likely state sequence \( S \) that best explains a given observation sequence \( o_{1:T} \), i.e., maximize \( P(S | o_{1:T}, \lambda) \), which is commonly determined by the Viterbi algorithm [37]. The time and memory complexity for training an HMM with \( N \) states according to Baum-Welch algorithm is \( \mathcal{O}(N^2T) \) and \( \mathcal{O}(NT) \) respectively, for a sequence of length \( T \) symbols.

Unlike algorithms for sequential data, which can directly learn from data streams, standard machine learning algorithms (e.g., K-Nearest Neighbor, Artificial Neural Networks and Support Vector Machines) require fixed-size numeric feature vectors as inputs. Therefore, a mapping from the system call traces into such feature vectors is required to allow the application of various machine
learning algorithms. Traditional data representation for text categorization or
document classification involves the term vectors or (bag of words), where each
document is represented by a vector of terms or words. The term vector is
a mapping from the document space to a fixed-size vector whose entries are
nonzero if the corresponding term appears in the document and zero otherwise.
Each term in the vector is typically weighted using the term frequency \( (tf) \) or
the term frequency-inverse document frequency \( (tf.idf) \) [38].

In host-based anomaly detection systems, the term vector or the bag of
system calls has been used to train one-class Naive Bayes algorithm and K-
Means clustering for anomaly detection as well as two-class classifiers, such as
decision tree, Naive Bayes, SVM and Logistic Regression for misuse detection
[17]. Chen et al. compared the performance of the support vector machine
(SVM) classifier to that of an artificial neural network (ANN) classifiers, both
trained using the term vector representation of system call traces [16]. They
showed that the detection accuracy of SVM was superior to ANN and the results
improved when the term vector is weighted by the \( tf.idf \) instead of just by \( tf \)
[16]. However, this data representation discards the temporal order among the
system calls in a trace.

In this work, we focus on building a multiple-detector anomaly detection
system based on the combination of these well-known and diverse detectors:
STIDE, HMM, and OCSVM. As described in Section 4.2, the OCSVM is trained
using a Gaussian kernel with a \( tf.idf \) weighted term vectors.

Recently, Creecy and Hu proposed a “semantic” feature extraction technique
from system call traces [39]. The basic idea is to create feature vectors of phrases
of different length (up to five in their experiments) comprising the frequency of
n-grams \( (n = 1, \ldots, 5) \) that appear in the training traces. As an anomaly
detector, the authors proposed to use the Extreme Language Machine (ELM),
which is a generalized single-hidden-layer feed-forward networks [40]. In fact,
ELM randomly chooses the input weights and analytically determines the output
weights of the feed-forward network, and hence requires less human interventions
and runs faster than conventional neural networks. However, it not clear how
the authors have trained the ELM technique using the normal traces only. They
showed that their proposed approach yielded the best detection accuracy on the
ADFA-LD dataset. We have compared the results of our system with their
results as shown in Section 4.3.

3. Iterative Boolean Combination of Detectors in the ROC Space

The iterative Boolean combination (IBC) technique summarized and further
clarified in this section has been proposed to combine detectors’ decisions in the
ROC space for an improved accuracy [20].

A crisp detector outputs a class label (e.g., normal or anomaly) while a soft
detector such as STIDE (using Hamming distance) or HMM assigns scores to the
input samples, which can be converted to a crisp detector by setting a decision
threshold on the scores. Given the responses of a crisp detector on a validation
set, the true positive rate \( (tpr) \) is the proportion of positives correctly classified
over the total number of positive samples. The false positive rate \( (fpr) \) is the proportion of negatives incorrectly classified over the total number of negative samples. The positive (or target) class is typically the class of interest, which is the anomalous class for an ADS.

A receiver operating characteristic (ROC) curve is a plot of \( tpr \) against \( fpr \) [41]. A crisp detector produces a single data point in the ROC plane, while a soft detector produces a ROC curve by varying the decision thresholds. In practice, an empirical ROC plot is obtained by connecting the observed \( (tpr, fpr) \) pairs of a soft detector at each decision threshold. With a finite number of decision thresholds, an empirical ROC plot is a step-like function which approaches a true curve as the number of samples, and hence the number of decision thresholds, approaches infinity.

The ROC curves have originally been introduced in signal detection theory to visualize the performance of detectors and select optimal operational points, without committing to a single decision threshold. ROC analysis presents the detectors performance across the entire range of class distribution and error costs. A random guess would give a point along the diagonal line joining the left bottom point \((0, 0)\) to the top right one \((1, 1)\), which is also called diagonal of chance. For equal prior probability and cost of errors, the optimal decision threshold corresponds to the vertex that is closest to the upper-left corner \((0, 1)\) of the ROC plane. This point can be graphically located on the ROC curve by moving a line with a slope of one (called iso-performance line), i.e., parallel to the diagonal of chance, until it touches the ROC curve.

In many practical cases, such as in intrusion detection systems, the prior probability of intrusions and misclassification costs (the cost of a false alarm versus the cost of missing an intrusion) are critical to evaluate the overall system performance. These information must be considered while analyzing the ROC curve and selecting the optimum operating points. The slope of iso-performance lines could be adjusted to account for the prior probability or the costs ratio [41, 42]. Another interesting technique applies the decision tree to provide expected cost metrics that account for the misclassification costs ratio at different prior probabilities intervals [19]. Our combination technique in the ROC space is based on Boolean combination of decisions form various thresholds (on the same or different ROC curves), independently from the prior probabilities or cost of errors. This allows for the application of techniques that account for the prior probabilities and costs of errors (e.g., [42, 19]) while selecting the optimal operating point to be readily applied on the combination results provided by our technique. As described in the rest of this section, an IDS based on the proposed combination technique is capable of changing the operating point during system operations to adapt to changes in prior probabilities and costs of errors (which may happen after deployment to operations).

The area under the ROC curve (AUC) has been proposed as more robust (global) measure for evaluation and selection of classifiers than accuracy or conversely error rate [43]. In fact, the accuracy depends on the decision threshold, and assume fixed distributions for both positive and negative classes. The AUC however is the average of the \( tpr \) over all values of the \( fpr \) (independently of
decision thresholds and prior class distributions). The AUC assesses ranking in terms of class separation, it evaluates how well a classifier is able to sort its predictions according to the confidence they are assigned. For instance, with an $AUC = 1$ all positives are ranked higher than negatives indicating a perfect discrimination between normal and anomalous classes, while a random classifier has an $AUC = 0.5$.

In some cases however the ROC curves may cross, and hence detectors providing higher overall AUC values may perform worse than those providing lower AUC values, in a specific region of ROC space [44]. In such cases, the partial area under a specific region of the ROC curve (for instance the area under the curve between $fpr = 0$ and $fpr = 0.1$), could be more useful for comparison than the area under the entire ROC curve [44]. However, if the AUC or the partial AUC values are not significantly different, the shape of the curves might need to be looked at.

In any case, any attempt to summarize a ROC curve into a single number leads to information loss, such as those incorporated in the prior probability and misclassification costs, which reduces the system adaptability in practice. Therefore, in our experimental evaluation we present the entire ROC curves for comparison and we mainly compare the $tpr$ of different detectors for specific $fpr$ values (in particular for $fpr = 0$). In addition, we provide the AUC value as a global metric of detectors’ performance. Note that, for the combined systems, the AUC is the area under the final composite ROC convex hull (as described next).

An important concept in ROC analysis is the ROC curves Convex Hull (ROCCH), which is the piece-wise outer envelope connecting only superior points in the ROC space [41, 42]. Given two operating points, say $a$ and $b$, in the ROC space, $a$ is defined as superior to $b$ if $fpr_a \leq fpr_b$ and $tpr_a \geq tpr_b$. A detector is therefore potentially optimal if it lies on the convex hull of the set of points in the ROC space. The ROCCH has been proposed for combination of detectors, based on a simple interpolation between their responses [41, 42, 45]. In practice, this is achieved by randomly alternating detectors responses proportionately between the two corresponding vertices of the line segment on the convex hull where the desired operational point lies. For example, given the ROC curves of two detectors $D_1$ and $D_2$ presented in Figure 1a, assume that the maximum tolerated false alarm rate is 10% ($fpr \leq 0.1$). Under such design constraints, the highest detection performance ($tpr = 25\%$) is achieved by detector $D_2$ at point $d$.

However by applying the ROC combination, the detection performance could be increased to point $c$, which has a $tpr = 35\%$. Although, the desired operating point $c$ does not lie on any ROC curves of the given detectors, which, in reality, means that there is no threshold $t$ that can be applied to output of $D_1$ or $D_2$ to provide the performance achieved by the point $c$, it can be achieved by taking the responses from $a$ and $b$ proportionally to their relative positions with reference to $c$ on the line segment $ab$. In our example, the ratio of the line segment $ac$ to $ab$ is about 0.3, therefore by randomly taking the decisions from
Algorithm 1: $BC_{ALL}(T_a, T_b, labels)$

**input**: Thresholds of two ROC curves, $T_a$ and $T_b$ and labels

**output**: ROCCH and fused responses ($R$) of combined curves; each point results from two thresholds combined with a Boolean function ($bf$)

1. let $m \leftarrow$ number of distinct thresholds in $T_a$;
2. let $n \leftarrow$ number of distinct thresholds in $T_b$;
3. allocate an array $F[2, m \times n]$ // temporary fusions;
4. $BooleanFunctions \leftarrow \{a \land b, \neg a \land b, a \land \neg b, \neg(a \land b), a \lor b, \neg a \lor b, a \lor \neg b, \neg(a \lor b), a \oplus b, a \equiv b\}$;
5. compute ROCCH old of the original curves;
6. for each $bf \in BooleanFunctions$ do
   7. for $i \leftarrow 1$ to $m$ do
      8. $R_a \leftarrow (T_a \geq T_a_i)$ // threshold to response;
      9. for $j \leftarrow 1$ to $n$ do
         10. $R_b \leftarrow (T_b \geq T_b_j)$;
         11. $R_c \leftarrow bf(R_a, R_b)$ // fuse responses;
         12. compute $(tpr, fpr)$ using $R_c$ and labels;
         13. push $(tpr, fpr)$ onto $F$;
      14. compute ROCCH new of $F$;
      15. push responses of points above ROCCH old into $R$;
      16. store their thresholds and Boolean functions to be used during operations;
      17. $ROCCH_{new} \leftarrow ROCCH_{old}$ // Update ROCCH;
7. return $ROCCH_{new}, R$

Point $b$ at a rate of 30% and from point $a$ at a rate of 70% ($1 - 0.3 = 0.7$) the point $c$ is realized; a gain of 10% in true positive rate at the same $fpr = 0.1$.

By selecting only superior detectors, the ROCCH combination discards responses from inferior detectors which may diverse information for an improved performance [20, 19]. As illustrated in the following examples, the Boolean combination of two detectors can produce new operating points in the ROC space, yielding superior performance than the ROCCH of the original curves [20].

The algorithm for Boolean combination ($BC_{ALL}$) of two ROC curves is presented in Algorithm 1, and illustrated in Figure 1. It inputs a pair of ROC curves defined by their decision thresholds and the labels for a validation set. For each of the ten Boolean functions (see line 4 of Algorithm 1), $BC_{ALL}$ combines the responses (on the validation set) of each threshold from the first curve with the responses of each threshold from the second (as illustrated in the right part of Figure 1b). The results of all combination are then mapped to points ($fpr, tpr$) in the ROC space (as illustrated in the left part of Figure 1b). The thresholds (from each curve) of combined points that are superior to the ROCCH of original curves are then stored in a database ($R$) along with their corresponding Boolean functions (as illustrated in Figure 1c). The ROCCH is then updated to include the new emerging points. The outputs are the vertices of the final ROCCH, where each point is the results of one thresholds from each ROC curve combined with the corresponding Boolean function, which are stored in ($R$) and applied
Algorithm 2: $BCM_{ALL}(T_1, T_2, \ldots, T_K, labels)$: Boolean Combination of Multiple ROC curves

input : Thresholds of $K$ ROC curves $[T_1, \ldots, T_K]$ and $labels$
output : ROCCH of combined curves; each point is the result of combination of two selected curves
1. $[ROCCH_{1:2}, R_{1:2}] = BCM_{ALL}(T_1, T_2, labels)$ // combine the first two ROC curves;
2. for $k \leftarrow 3$ to $K$
   // combine the responses of the previous combination with those of the following ROC curve;
3. $[ROCCH_{1:k}, R_{1:k}] = BCM_{ALL}(R_{1:k-1}, T_k, labels)$
4. store selected thresholds and Boolean functions;
5. return $ROCCH_{1:K}, R_{1:K}$

Algorithm 3: $IBC(T_1, T_2, \ldots, T_K, labels)$: Iterative Boolean Combination

input : Thresholds of $K$ ROC curves $[T_1, \ldots, T_K]$ and $labels$
output : ROCCH of combined curves; each point is the result of a composite combination
1. set maxiter and tol // number of iterations allowed and tolerance between AUCH values;
2. iter $\leftarrow 1$;
3. $[ROCCH_{iter}, R_{iter}] = BCM_{ALL}([T_1, T_2, \ldots, T_K], labels)$;
4. for iter $\leftarrow 2$ to maxiter do
5.   $[ROCCH_{iter}, R_{iter}] = BCM_{ALL}([R_{iter-1}, T_1, T_2, \ldots, T_K], labels)$;
6.   if $(AUCH_{iter} \leq AUCH_{iter-1} + tol)$ then
7.     return // no significant improvement;
8. store selected thresholds and fusion functions for each iteration;
9. return $ROCCH_{iter}, R_{iter}$

during operations.

The example presented in Figure 1 illustrates the improvement in performance achieved by the Boolean combination in the ROC space. For the same constraints on the maximum tolerable false positive value, $fpr = 10\%$, the $BC_{ALL}$ is now able to achieve a $tpr = 50\%$ (point $e$), i.e., a $15\%$ increase in detection accuracy over point $d$ (achieved by $CH_1$, the ROCCH of the original detectors). Again the new operational point ($e$), which corresponds to $fpr = 10\%$, is located between points $c_{12}$ and $c_{13}$ of the new composite ROCCH $CH_2$, and can be achieved by interpolation of their responses (by randomly taking the decisions from point $c_{12}$ at a rate of about $30\%$ and from point $c_{13}$ at a rate of about $70\%$). The point $c_{12}$ is achieved by negating the decisions of detector $D_1$ at threshold $t_1$ and AND'ing them with the decisions of detector $D_2$ at threshold $t_2$. Similarly, the point $c_{13}$ is achieved by the following Boolean combination rules retrieved from the database $R$: $c_{13} = (D_1, t_2) \lor (D_2, t_1)$. It is important to notice that the $BC_{ALL}$ algorithm allows the system to operate on point $c_{11}$,
Figure 1: Example of the steps involved with the Boolean combination of two detectors ($D_1$ and $D_2$) in the ROC space (according to Algorithm 1).

which has a true positive rate similar to that of point $c$ ($tpr = 35\%$), however with zero false alarms ($fpr = 0\%$).

Given $K$ detectors for combination, the complete Iterative Boolean combination (IBC) algorithm, first proposed in [20], is described in Algorithm 3. The IBC starts with the combination of a pair of detectors according to $BC_{ALL}$. The outputs of this combination are the selected vertexes, which have emerged above the ROCCH of the original curves, that are stored in the database $R$ along with their corresponding thresholds and Boolean functions (as shown in Figure 1c). The responses from these vertexes are then combined with those of the third detector as described in Algorithm 2. The results from this combination are the
vertexes that emerged above the previous ROCCH of the previous combination. These vertexes are then appended to the database along with their thresholds and Boolean functions, and so on until all detectors are combined.

As described in Algorithm 3, when all $K$ detectors have been combined (first iteration), IBC re-combines the resulting responses of the emerged vertexes, lying on the facet of the ROCCH, with each individual ROC curve, while selecting and saving the emerging vertexes and their Boolean combinations. This step is repeated iteratively until a maximum iteration number is reached or the ROCCH improvement and hence the increase in AUC values drops below a given threshold. This iterative procedure can further improve the overall performance of the combination and typically requires few (less than ten) iterations to converge. The results of iterative Boolean combination of $K$ ROC curves are the selected vertexes on the facet of the final ROCCH, where each vertex represents an ensemble of crisp detectors and Boolean functions selected from the original detectors according to specific thresholds. Although the iterative process does not necessarily provide an optimal set of combinations, its time complexity is linear in the number of detectors. To ensure that the method provides equal or better performance, since the ROC curves of the original detectors are always included to the ROC spaces, the performance of combination is guaranteed to be greater than or equal to that of the best detector.

The Boolean combination of detectors responses in the ROC space requires no assumption about the independence of detectors, in contrast with other fusion functions such as sum, product or averaging [46, 47]. The normalization of the output scores from heterogeneous detectors is also not required, because ROC curves are invariant to monotonic transformation of decision thresholds [48]. More importantly, since the IBC aims at maximizing the ROCCH of combinations over all decision thresholds provided by the combined detectors, it allows to adapt to changes in prior probabilities and cost of errors during system operation. When the operating point changes, different vertices on the ROCCH will be selected and hence different thresholds and Boolean functions will be activated to accommodate this change, without the need for retraining new detectors. This is an important property for ADSs, since the prior class distributions are highly imbalanced and misclassification costs are different and both may change over time; hence the operating point may change during operation. The next section describes how the iterative Boolean combination is applied to form ADSs based on multiple homogeneous or heterogeneous detectors trained on normal system call traces.

3.1. Illustrative Example

The following example provides further insight into why the Boolean combination in the ROC space can improve the detection accuracy while reducing the false alarm rate. Suppose that you have been provided by a validation set comprising 20 labeled system call sequences (5 anomalous and 15 normal) and the scores of two models $M_1$ and $M_2$ (have been previously trained on normal sequences) for each of these sequences, as shown in Table 1. The ROC curves of models $M_1$ and $M_2$ and their AUC values are illustrated in Figure 2. When
Table 1: Scores provided by models $M_1$ and $M_2$ to the sequences of system calls ($S_1, \ldots, S_{20}$) in the validation set, labeled as normal (0) or anomalous (1), followed by the decision of the selected crisp detectors ($A, B, C, D$) and their Boolean combinations.

<table>
<thead>
<tr>
<th>Seq. Label</th>
<th>Model Scores</th>
<th>Selected Detector Predictions</th>
<th>Boolean Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M_1$</td>
<td>$M_2$</td>
<td></td>
</tr>
<tr>
<td>$S_{1}$</td>
<td>1</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>$S_{2}$</td>
<td>1</td>
<td>0.59</td>
<td>0.95</td>
</tr>
<tr>
<td>$S_{3}$</td>
<td>1</td>
<td>0.80</td>
<td>0.95</td>
</tr>
<tr>
<td>$S_{4}$</td>
<td>1</td>
<td>0.58</td>
<td>0.45</td>
</tr>
<tr>
<td>$S_{5}$</td>
<td>1</td>
<td>0.40</td>
<td>0.35</td>
</tr>
<tr>
<td>$S_{6}$</td>
<td>0</td>
<td>0.60</td>
<td>0.91</td>
</tr>
<tr>
<td>$S_{7}$</td>
<td>0</td>
<td>0.50</td>
<td>0.59</td>
</tr>
<tr>
<td>$S_{8}$</td>
<td>0</td>
<td>0.70</td>
<td>0.99</td>
</tr>
<tr>
<td>$S_{9}$</td>
<td>0</td>
<td>0.45</td>
<td>0.40</td>
</tr>
<tr>
<td>$S_{10}$</td>
<td>0</td>
<td>0.52</td>
<td>0.80</td>
</tr>
<tr>
<td>$S_{11}$</td>
<td>0</td>
<td>0.53</td>
<td>0.70</td>
</tr>
<tr>
<td>$S_{12}$</td>
<td>0</td>
<td>0.55</td>
<td>0.80</td>
</tr>
<tr>
<td>$S_{13}$</td>
<td>0</td>
<td>0.40</td>
<td>0.80</td>
</tr>
<tr>
<td>$S_{14}$</td>
<td>0</td>
<td>0.42</td>
<td>0.70</td>
</tr>
<tr>
<td>$S_{15}$</td>
<td>0</td>
<td>0.40</td>
<td>0.60</td>
</tr>
<tr>
<td>$S_{16}$</td>
<td>0</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>$S_{17}$</td>
<td>0</td>
<td>0.50</td>
<td>0.40</td>
</tr>
<tr>
<td>$S_{18}$</td>
<td>0</td>
<td>0.20</td>
<td>0.30</td>
</tr>
<tr>
<td>$S_{19}$</td>
<td>0</td>
<td>0.85</td>
<td>0.99</td>
</tr>
<tr>
<td>$S_{20}$</td>
<td>0</td>
<td>0.89</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Threshold (see Figure 2): $T_1$, $T_2$, $T_3$, $T_4$

False positive rate: 0.27, 0.20, 0.27, 0.93, 0.07, 0.00, 0.34
True positive rate: 0.80, 0.00, 0.40, 0.80, 0.80, 0.40, 1.00

Figure 2: Illustration of the improvement achieved by Boolean combination of the crisp detectors selected from models $M_1$ and $M_2$ in the ROC space.

the output scores of $M_1$ and $M_2$ (which are also the unique thresholds of their ROC curves) and the sequence labels are input into Algorithm 1, the outputs are the ROCCH of combinations, the selected crisp detectors (thresholds on
the models), and the Boolean functions, as shown in Figure 2. The selected crisp detectors are denoted by letter A, to D on the ROC curves and $T_2$, $T_3$, $T_1$ and $T_4$ are their corresponding thresholds. These thresholds as well as the true and false positive rates of each detector are also shown in the lower part of Table 1. For instance, the crisp detector D is realized by setting a cut off point of $T_2 = 0.58$ on the scores of $M_1$. All scores that are greater to or equal to this threshold are considered anomalous and given a label of one; otherwise they are considered normal and labeled with zeros. The Boolean combinations of selected detectors are shown in Table 1 for each sequence, and their corresponding $fpr$ and $tpr$ are computed at the bottom of the table and mapped to ROC space in Figure 2, (denoted by E, F, and G). For instance, point F, which improves the detection rate from 20% to 40% while maintaining zero false alarm rate can be achieved, in practice, by negating the decisions of detector B and ANDing them with those of detector D. Any operation point between F and E can be realized by interpolation of responses, as described in previous section. The overall AUC of ROCCH of combination has increased to 0.95 compared to 0.69 of the original model $M_1$.

4. Experiments

The main objectives of our experiments are to evaluate and compare the performance of the proposed multiple-detector ADSs formed by the iterative Boolean combination of heterogeneous (STIDE, HMM, and OCSVM) detectors to that based on homogeneous Ensembles of HMMs (EoHMMs) using the modern and recently published system calls datasets: AFDA-LD for Linux [22] and CANALI-WD for Windows [23]. In addition, the performance of the multiple-detector ADS is compared to the ADS proposed in [39], presented in Section 2, which achieves the best results to date on the AFDA-LD dataset (described next).

4.1. Datasets

The normal system call traces are typically collected from the monitored host system in a secured environment, during normal system operation. These traces are assumed attack-free and used for training the anomaly detectors. The testing traces are the system calls collected from the same host while being under different kind of attacks. These attack traces comprise both normal and anomalous sequences for testing. Therefore, during testing, the whole attack trace or the collection of attack traces (an attack could result in more than one trace as shown in Table 2) is usually considered as one anomaly.

The UNM datasets have been commonly used for benchmarking ADSs based on system calls sequences for about 20 years now [2], due to the lack of publicly available datasets. However, neither the normal nor the attack traces represent the complexity of modern software application or the architecture or contempo-
Table 2: Normal versus Anomalous Traces in ADF A-LD and CANAI-WD Datasets

<table>
<thead>
<tr>
<th>Number of normal (benign) traces</th>
<th>Number of anomalous traces</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADF A-LD (Linux Datasets)</strong></td>
<td></td>
</tr>
<tr>
<td>training</td>
<td>833</td>
</tr>
<tr>
<td>testing</td>
<td>4373</td>
</tr>
<tr>
<td><strong>goodware</strong></td>
<td>629</td>
</tr>
<tr>
<td><strong>malware</strong></td>
<td>5855</td>
</tr>
<tr>
<td><strong>anubis-good</strong></td>
<td>36</td>
</tr>
<tr>
<td><strong>malware-test</strong></td>
<td>1200</td>
</tr>
</tbody>
</table>

Table 2 presents the number of normal and attack traces in the ADF A-LD dataset. The authors have divided the normal traces into 833 traces for training and 4373 for testing (a ratio of about 1:5), and used this normal data partitioning to training and evaluation of their proposed technique [22, 39].

In addition to the Linux system call datasets, we evaluated the proposed multi-detector approach using modern datasets of system call traces generated from several Windows machines. These datasets are provided by Canali et al. [23], and hence we refer to hereafter by CANALI-WD (CANALI Windows Datasets). In contrast to previously published system call datasets, which have limited anomalous traces, CANALI-WD provides a large collection of anomalous traces. As presented in Table 2, CANALI-WD contains 5,855 malware traces randomly extracted from Anubis\(^3\), including malwares such as botsnets, worms, dropper, Trojan horses. It also includes 1,200 traces of malwares (called malware-test) that have been collected on a different machine than that used by Anubis. On the other hand, CANALI-WD contains 629 traces of normal

---

\(^1\)http://www.cybersecurity.unsw.adfa.edu.au/ADF A IDS Datasets

\(^2\)http://www.metasploit.com

\(^3\)http://anubis.iseclab.org
execution (about 180 GB) collected from 10 different real-world machines, used by regular computer users (called goodware). The average length of these normal traces is about 1.2 million system calls, while the longest could reach 100 million system calls. Finally, it contains the traces of 36 normal application executed under Amubis (called anubis-good). Overall, the CANALI-WD dataset consists of a total of 1.5 billion system calls invoked by over 363,000 processes on different machines [23].

4.2. Experimental Protocol

For the ADF-A-LD dataset, we followed the experimental setup provided in [39]; therefore, the 833 normal traces are used for training STIDE, HMM, and OCSVM detectors. However, we hold out 1000 traces randomly selected from the 4373 normal traces and 20 attacks randomly selected from the 60 attacks for validation. This validation set is used to select the Window Size ($W$) for STIDE detector, the number of states for HMM, the kernel parameters for OCSVM and compute the Boolean combination and select the decision thresholds for the IBC technique. The remaining (3373) normal traces and (40) attack traces are only used for testing and benchmarking the detection performance of detectors. Since each attack consists of multiple traces (see Table 2), if an anomaly is detected in any of the traces belonging to the same attack, then the attack is considered successfully detected [39].

For CANALI-WD dataset, we used anubis-good traces and traces for nine out of the 10 machines in the goodware dataset (similar to the authors methodology [23]) to train the STIDE, HMM, and OCSVM detectors. In contrast with the authors however, where the malware traces were also used to build their models, we used the malware traces for evaluation. This is because we focus on anomaly detection where the anomaly detectors are only trained on normal system calls. The traces from the tenth machine in the goodware, the malware-test traces and malware traces are instead used to evaluate the models. Finally, a fraction of about 10% of normal and anomalous system call for testing is used as a validation set.

Training STIDE only requires the selection of the window size value. In our experiments we trained STIDE using $W = \{5, 10, 20\}$ on the normal traces provided for training, and used STIDE with Hamming distance to evaluate the score of a new test sequence. Since STIDE with $W = 5$ provided the best detection performance on both validation sets (as measured by its ROC curves), it was selected and considered for the combination with HMM using the IBC algorithm.

As described in Section 2, estimating the parameters of an HMM requires the specification of the number output symbols ($M$) and the number of hidden states $N$, as well as its topology. The number of output symbols is taken equal to the host system alphabet size ($M = 340$ unique symbols for ADF-A-LD and $M = 275$ for CANALI-WD). Since using a single HMM with a pre-specified number of states may have limited capabilities to capture the underlying structure of the data (as discussed in Section 2), therefore, different discrete-time
ergodic HMMs are trained with various $N = 10, 20, \ldots, 200$ values. The iterative Baum-Welch algorithm is used to estimate HMM parameters [49]. To reduce overfitting effects, the evaluation of the log-likelihood on the validation set is used as a stopping criterion. For each state value, the training process is repeated ten times using a different random initialization to avoid local minima, and the HMM that gives the highest log-likelihood value on the validation data is selected for the IBC combination. The FB algorithm is used to evaluate the performance of each trained HMM, which should assign significantly lower likelihood values to anomalous sequences than to normal ones.

The OCSVM detector is trained using the term vectors with $tfidf$ weights extracted from the (normal) training traces. The training of OCSVM is done using LIBSVM\cite{4}, a commonly used library for support vector machines. In our experiments, we have trained and compared the performance of OCSVM using a Gaussian or RBF (radial basis function) kernel: 
\[
K(v_i, v_j) = \exp\left(-\frac{|v_i - v_j|^2}{2\sigma^2}\right)
\]
where $v_i$ and $v_j$ are the feature vectors and $\sigma^2$ denotes the variance. The values that provided the best performance on the validation sets are $\sigma = 0.001$ for ADF A-LD dataset and $\sigma = 0.00001$ for CANALI-WD dataset.

For each dataset, the detector with best performing parameter settings on the validation sets, i.e., STIDE ($W = 5$), HMMs ($N = 200$), and OCSVMs ($\sigma = 0.001$ for ADF A-LD, and $\sigma = 0.00001$ for CANALI-WD dataset) are selected for combination according to IBC in order to construct the heterogeneous multiple-detector ADS.

To construct and select the EoHMMs, we applied the IBC algorithm on the validation set to combine the ROC curve of HMM with $N = 200$ (since it provided the best ROC curve on validation) with the ROC curve of each of the remaining HMMs. The combination with an HMM trained with $N = 30$ on ADF A-LD and $N = 10$ on CANALI-LD provided the best AUC on the validation set, and hence selected for comparison.

### 4.3. Experimental Results

As described in Section 3, we evaluate the performance by comparing the $tpr$ achieved by each technique at zero false alarm rate ($fpr = 0$). We also present the entire ROC curves and composite ROCCH of combined detectors for comparison at different operating points. In addition, we provide the AUC value as a global metric of detectors’ performance. Figure 3 presents the results obtained on ADF A-LD dataset, while Figure 4 presents those obtained on CANALI-WD dataset.

For the ADF A-LD dataset, the ROC curves of the best performing ADSs that are based on the proposed multiple-detector and on the EoHMMs are presented in Figure 3a and 3b. Figure 3a presents the complete ROC curves achieved by the multiple-detector ADS for the combination of the selected detectors STIDE with $W = 5$, HMMs with $N = 200$, and OCSVMs with $\sigma = 0.001$ and

\footnote{http://www.csie.ntu.edu.tw/~cjlin/libsvm}
Figure 3: ADF-A-LD: ROC curves and AUC results of our multiple-detector ADS (a), compared to the EoHMMs based ADS (b), to those obtained by Creech and Hu [2013b], and to each individual detector.
compares them to that obtained by the system proposed in [39], and to each individual detector (before combination).

As illustrated by the arrows on Figure 3a, at an operating point of zero false alarm rate $fpr = 0$, our system achieves about 90% true positive rate ($tpr = 0.9$) compared to less than 10% true positive rate ($tpr \leq 0.1$) obtained by the ADS proposed by Creec'h and Hu [39] or by STIDE detector. The ROC curve achieved by the multiple-detector ADS completely dominates all other curves, in particular it dominates that obtained in [39] for all decision thresholds. This improved detection accuracy is also shown in the AUC values in the legend of the figure, where the proposed ADS combining the three detectors, $IBC(D_1, D_2, D_3)$, achieves the highest overall AUC value of 0.993.

Figure 3b presents the ROC curves of the EoHMMs based ADS, achieved by the IBC of HMM trained with $N = 200$ to that trained with $N = 30$, which is the best performing EoHMMs on the validation sets as described in Section 4.2. The ROC curve obtained by the system proposed in [39] is also presented for comparison. The results in this figure indicate better ROC curves and AUC performance than those of Creec'h and Hu’s system, but worse than those obtained by the proposed multiple-detector ADS in Figure 3a. In particular, at an operating point of zero false alarm rate the EoHMMs achieves a true positive rate of about 15% compared to 90% obtained by the heterogeneous multiple-detector ADS of Figure 3a.

As shown in Figure 4, the results obtained using the Windows system call dataset, CANALI-WD, confirm the results obtained on the Linux system call dataset (ADFA-LD). The true positive rate at zero false alarm rate achieved by the multiple-detector is about 80% compared to 30% achieved by the OCSVM (see arrows on Figure 4a) and compared to 64% achieved with EoHMMs (see arrows on Figure 4b). These results indicate an increase of about 166% in the $tpr$ over that of OCSVM detector and an increase of about 25% in the $tpr$ over that of EoHMMs. The area under the ROC curve obtained by our multiple-detector, $IBC$(STIDE with $W = 5$, HMM with $N = 200$, and OCSVMs with $sigma = 0.00001$) is $AUC = 0.984$, as shown in Figure 4a, while that of the EoHMMs, $IBC$ (HMM with $N = 200$ and HMM with $N = 10$), is $AUC = 0.958$ (Figure 4b).

Overall, the results on both ADFA-LD (Figure 3) and CANALI-WD (Figure 4) datasets show that the proposed multiple-detector ADS consistently outperforms each of the individual detectors as well as the ADS based on EoHMMs. In addition, on the ADFA-LD dataset, it outperforms the ADS proposed by the creator of the dataset [39]. The results presented in Figure 3 for the Linux (ADFA-LD) dataset indicate a 500% increase in the $tpr$ at $fpr = 0$ over the best results achieved by the existing techniques, and an increase of 25% in the $tpr$ is shown in Figure 4a for the Windows (CANALI-WD) dataset. These results confirm that the iterative Boolean combination is able to exploit the diverse and complementary decision information provided by the combination of heterogeneous detectors (STIDE, HMM, and OCSVM), to achieve better results than combining homogeneous ensembles of the same detectors (such as EoHMMs).
We kept the number of combined detectors intentionally low (two or three detectors only) to keep a fair companion with other ADSs that are based on one detector only. In practice, the only restriction on including other detectors would be an increased system complexity beyond the design constraints. However, as discussed in Section 4.4 the overhead of the IBC during the operational phase is negligible with reference to those of the combined detectors. Our combination technique in the ROC space is based on Boolean combination of decisions form various thresholds independently from the prior probabilities or cost of errors. Techniques that account for changes in the prior probabilities and costs of errors [42, 19] could be used to select the optimal operating point. More importantly, our technique is capable of changing the operating point during system operations to adapt to changes in the prior probability or cost of errors by activating a different set of decision thresholds and Boolean functions. Future work include investigating a larger number of complimentary detectors for combination in host-based ADS using system calls. Combining detectors working on different features according to the IBC is expected to further increase the detection accuracy while decreasing the false alarm rates. In addition, it will also make the ADS more robust and resilient to evasion or adversarial attacks as detailed in Section 5.

4.4. Time and Memory Complexity Analysis

The IBC algorithm is very efficient in practice. The space of all Boolean combination is doubly exponential and finding the optimal set of combination is practically infeasible even for a small number of detectors. For \( n \) binary detectors there are \( 2^n \) possible outcomes that can be combined in \( 2^{2n} \) Boolean combinations.

During the design phase, it reduces the time complexity from \( \mathcal{O}(2^n) \) to the order of \( \mathcal{O}(n^2) \) Boolean combinations, in the worst-case. A more detailed analysis shows that the IBC algorithm requires fewer than \( \mathcal{O}(n^2) \) Boolean operations due to its sequential nature. Given \( K \) ROC curves (from \( K \) soft detectors) each with \( T_k \) decision thresholds (i.e., each providing \( T_k \) crisp detectors), and let \( n = T_1 + T_2 + \ldots + T_k \) be the total number of crisp detectors. The worst-case time complexity required by the IBC algorithm to apply the ten Boolean functions (to each decision threshold from each ROC curve) over \( n_{\text{iter}} \) iterations is of the order of \( \mathcal{O}(n_{\text{iter}}T_1T_2) \), where \( T_1,T_2 \ll n^2 \). This is because, after combining the first two ROC curves \( \mathcal{O}(T_1T_2) \), the number of emerging combinations that push the composite ROCCH over that of the original curves, is typically orders of magnitude smaller than \( T_1T_2 \). For example, Figure 2 shows that only three combinations improved the original ROCCH. These points are then combined with the third curve using \( \mathcal{O}(T_3) \) Boolean operations, and so on. Similarly, the time complexity required by each subsequent iteration are reduced by an order of magnitude with respect to that of the first iteration. The memory complexity required to store the temporary results \((tpr, fpr)\) of each Boolean function is \( \mathcal{O}(T_1T_2) \). In our experiments, the IBC algorithm required an average of three iterations \((n_{\text{iter}} = 3)\), and the average time for computing the Boolean combination of STIDE, HMM and OCSVM was about 5 seconds.

20
Figure 4: CANALI-WD: ROC curves and AUC results of our multiple-detector ADS compared to the EoHMMs based ADS, and to each individual detector.
During operations, the IBC requires the application of the selected Boolean functions to the output of the combined detectors at the selected thresholds. Therefore, the overhead of the combinations on the overall time complexity of the multiple-detector ADS is negligible with reference to that of the original detectors. Since the sequence of system calls is input in parallel to all detectors, the overall time required to output a response is, in fact, the time required by the slowest detector to provide its response, which also depends on the choice of detectors parameters.

The worst-case time complexity of the Forward-Backward algorithm to evaluate a sequence of $W$ observations with an $N$ states HMM is of the order of $O(N^2W)$ [37]. Selecting an HMM with small $N$ values is therefore more efficient in practice, since its time complexity grows quadratically with the number of states, and linearly with the detector window size. On the other hand, the time complexity of STIDE using Hamming distance is $O(W(R_A S_u + 1))$, where $R_A$ is the rate of anomalous to normal sequences, and $S_u$ is the number of unique sequences of length $W$ in the normal database [24]. The additional computational complexity required by STIDE using Hamming distance over STIDE as crisp detector stems from the fact that if a test sequence is not exactly found in the normal database STIDE must compute the Hamming distance between the test sequence and all $(S_u)$ sequences in the database [24]. Although the time complexity of STIDE seems to grow linearly with the detector window size $W$, in practice, the number of unique sequences $S_u$ could increase exponentially with the large $W$ values (also depending on $R_A$). Consequently, a smaller window size should be always favored to reduce the response time of STIDE (and also its storage requirements), provided that it is large enough to detect the minimal foreign anomalous sequences [50]. For OCSVMS, the input sequence is first transformed into a fixed size vector of length $M$ (system call alphabet size) and then input to the detector. The worst case time complexity for classifying a vector of size $M$ system calls is $O(M)$. The selection of the detectors and their parameters could be considered during the design phase to minimize the response time if very low latency is required by the monitored environment.

4.5. Threats to Validity

We identify the following key threats to validity. First, the confidence in the Boolean combination and thresholds selected by IBC during the design phase relies on the validation set. If the validation set provided for designing the ADS is not representative, the resulting combination may not provide the expected performance during operations. However, the ROCCH convex hull of both detectors is guaranteed to be the lower bound on the operational performance of IBC, since it is always the starting point of this combination. This means that the performance provided by the IBC algorithm is equal to or greater than that of the ROC convex hull of both detectors.

A threat to internal validity exists in the implementation of anomaly detection techniques (STIDE, HMMs, OCSVMS) and the Boolean combination techniques, as well as in conducting the experiments for anomaly detection. We have mitigated this threat by a manual verification of their outputs.
5. Discussion of Evasion and Adversarial Attacks

In the section we limit our discussions to the related works that have not been discussed in Section 2, in particular we focus on evasion and adversarial attacks against system call ADSs. In general, all intrusion detection systems are susceptible to evasion or adversarial attacks. As soon as IDSs are deployed, they may become target of adversarial attacks that try to evade, undermine or mislead their detection capabilities [51].

Mimicry attacks were among the earliest attempts toward defeating host-based ADSs that only monitor the temporal order of system calls. Wagner et al. proposed that it is possible to craft sequences of system calls which appear normal to the ADS (hence they will not be detected) while exploiting some vulnerabilities in the monitored process [27, 52]. The authors proposed replacing the foreign system calls, which do not belong to the normal process behavior (and can be easily detected), with one or multiple nullified system calls that belong to the normal system behavior. Nullified system calls are legitimate calls but have no effect (similar to no operation “no-op” system call) since their return values and the parameters are ignored. This form of mimicry attacks allow an attacker to embed the malicious sequence of system calls (necessary to run the exploit) within the sequences that belong to the normal process behavior, by careful substitution and padding of nullified calls. The authors formulate the generation of mimicry attack sequence as a finite-state automata intersection and showed that an initial detectable exploit of eight system calls can be transformed into a mimicry attack of length 100 system calls.

In our opinion, the presence of mimicry attacks does not diminish the need for anomaly detection systems based on system call sequences. In fact, it is quite the opposite. It encourages researchers to combine models of system call sequences with other models built from additional system artifacts such as system call arguments [53, 54, 55, 56], memory and call stack information [57, 58, 26], and function calls and other user-space information [59, 60]. The long-term goal is to work towards an anomaly detection infrastructure with multiple layers of security, as further detailed below. With this in mind, system call based techniques that can reduce false alarms while keeping a decent level of accuracy such as the one we have introduced in this paper, should contribute to building such a holistic solution.

Another form of evasion attack relies on crafting attacks sequences that exploit specific weaknesses in the detection coverage of the sequence matching anomaly detectors that are based on a sliding-window, such as STIDE [61]. An example of such blind regions is provided in [61] showing that the detector window size ($W$) of STIDE must be at least equal to the smallest anomalous sequence of the attack to be visible for the detector. Otherwise, the window of STIDE detector will slide on the subsequences of the anomalous sequence (which are all normal), without being able to discover that the whole sequence is anomalous. This blind region issue only affects binary or crisp window-based detectors that produce a class label (normal or anomalous) for a given sequence. In contrast, probabilistic and sequential detectors have no blind regions and
will able to detect the anomalous sequence even if its length is greater than the detector window size [61]. Our multiple-detector ADS is resilient to such evasion attacks, because the attacker must now simultaneously predicts two (or more) thresholds as well as the applied Boolean functions (which are difficult to guess) to remain undetected.

An attacker could also attempt to predict the threshold that raises the alarm in order to make his attack go undetected (under the radar) [50]. As described in Section 3, any ADS provides a trade-off between true and false positives. In order to reduce the number false positives (i.e., smoothen the false alarms), several researchers used another temporal threshold on a recent history of events. Instead of raising an alarm when one subsequence is detected as anomalous, the test sequence is only signaled as an attack if the number of anomalous subsequences within a recent time window exceeded a given threshold. This has been called a *locality frame* in [2], since the anomaly signal is computed from the number of mismatches occurring in a temporally local region. However, this second threshold is typically set to arbitrary values and opens the possibility of crafting attacks that remain under the threshold value, by producing the spreading the anomalous sequences over a period of time longer than the locality frame. Our approach does not rely on smoothing thresholds to the reduce false alarms, but on the on the Boolean combination of detectors, and hence this kind of attack is not applicable.

An alternative type of evasion attacks against the control-flow relies on exploiting the system call arguments to evade the detection of ADSs monitoring system call sequences. If an attacker is able to launch the attack by exploiting the arguments of system calls without tempering the normal order of system calls, then it may go undetected by the ADS since the arguments are not monitored [52]. Recent works included additional information about the system call arguments to defend against such attacks [53, 54, 55, 56]. However, these approaches have difficulties in deciding which legitimate argument value is really benign, when multiple legitimate values appear in the training phase [62]. Our approach allows to include additional detectors (specialized for examining different features such as system call arguments, return values and other information flow features) in a modular way. This is achieved by only recomputing and automatically selecting the Boolean combinations and decision thresholds that reduce the overall false alarm rate while increasing the true positive rate.

The anomaly detection techniques, described above, which try to defend against control-flow attacks using both the system call sequences (temporal order) or the system call arguments, have been called black-box detectors [38]. In contrast, the white-box detectors examine the program being monitored by statically analyzing the source code or binary files [63, 64, 65, 27]. Gao et al. coined the term gray-box for the anomaly detector that does not utilize static analysis of the program source code, but does extract additional runtime information from the monitored process when a system call is invoked, by looking for instance into the memory allocated for that process [58]. Sekar et al. proposed the first gray-box anomaly detector by including the program counter of the process with the system call number [26], while Feng et al. further incorporated
the return addresses on the call stack of the process when each system call is invoked [57]. Tandon and Chan coupled the system call arguments with their return values [56]. Gao et al. proposed an execution graph model that accepts sequences of system calls as well as the active function calls when each system call is invoked [59]. The previous approach has been further extended using the call stack information to provide more context for system call arguments, and introduced a metric that quantifies the degree to which system call arguments are unique to particular execution contexts [60].

These gray-box approaches made evasion and mimicry attacks harder, because the attack code will not be able to resume control after the execution of a system call. In fact, if the attacker attempts to regain the execution control by providing a return address on the stack, the ADS monitoring the return values would detect the presence of the attack. However, Kruegel et al. devised an approach that relies on corrupting the data in register contents or local variables to regain control of the program execution flow after a system call is completed [66]. The authors focused on demonstrating the ability of their symbolic execution technique to generate configurations to return the control to the attack code. However, several issues that need to be addressed before constructing such attacks against real-world applications were left open.

The main focus of the above approaches was mainly against code-injection attacks to compromised the host system. Chen et al. demonstrated other kind of attacks that do not modify the control flow of a program; instead, they exploit the (non-control) data-flow to take full control of the system [67]. The authors also demonstrated exploits against data-flow vulnerabilities by, for instance, using normal system calls to overwrite the password file and then elevate privileges [67]. Similar kind of attacks have been also applied to common web servers by targeting security-critical data, such as variables that store the user identification numbers corresponding to an FTP client and the directory that contains all allowable CGI scripts for a web server [68]. Some non-control data-flow attacks require no invocation of system calls, therefore the attacks will most likely evade detection by system-call based monitoring mechanisms. For instance, the persistent interposition attacks proposed by Parampalli et al. are based on injecting code that interposes on input/output operations, by modifying the data read or written by the victim, but leaving the control-flow and other system-call arguments unmodified. Although these persistent interposition attacks do not aim at compromising the system (e.g., by obtaining a root shell), they are powerful enough to steal credit card numbers and passwords or server’s private key, or alter emails [69]. These attacks do not manifest at the system call level, and hence are outside the scope of system call based ADS.

In practice, however, it may be difficult to launch an evasion or a mimicry attack without disrupting the order of the system calls. As shown in [70], the actions taken by the attacker before and while launching his attack (within the preamble phase), may produce deviations from the normal behavior of the monitored system that could be detected at the system call level, before the attacker proceeds to take full control of the system or perform other stealthy actions.
In summary, we think that the key problem of anomaly detection systems is the high rate of false alarms. An anomaly detector that generate an excessive number of false alarms is not useful, especially that an expensive and time consuming investigation is required to confirm or refute each alarm. Therefore, an ADS monitoring the temporal order of system calls that generates a small number of false alarms provides an important first line of defense. Attacks that have no manifestations at the system call sequence level could be detected with ADSs that rely on additional information about the system call arguments, return values, call stack, function calls, or other runtime and memory information, as described above. We strongly believe in a layered defense architecture that employs several independent defense strategies to create a more robust overall protection. An adversary is then forced to craft attacks that must conform to normal behavior of the system from various point of views, depending on several detection techniques and features. The modularity of the proposed multiple-detectors ADS provide an efficient and easy to apply solution because it operates at the decision level. It allows to include any additional detector (specialized to detect specific features) by only recomputing the Boolean combinations and decision thresholds (which is very efficient as described in Section 4.4). Our multiple-detector ADS is resilient to evasion attacks, because the attacker must now simultaneously predicts two (or more) thresholds and the applied Boolean functions. Furthermore, the operating point could be changed during system runtime, which activate different detectors, decision thresholds and Boolean functions. More importantly, the combination of two or more detectors compensates for the weaknesses of each individual detector. Exploiting the diversity of detectors that commit different errors is at the heart of multiple detector (or classifiers) systems.

6. Conclusions

This paper presents a multiple-detector ADS based on Boolean combination of responses from heterogeneous detectors in the ROC space. The experiments show that the proposed combination of decisions from STIDE, HMM, and OCSVM (which are among the most well investigated and best performing detectors for host-based ADS using system call sequences) consistently yields a significant increase in the detection accuracy over that of individual detectors, while reducing the false alarm rate. This gain in detection accuracy and reduction in false alarms are achieved without a considerable overhead. In fact, during system operation the time and resource requirements are those required for operating the selected individual detectors. This is due to the ability of the Boolean combination technique to efficiently exploit the diverse views provided by STIDE, HMM and OCSVM for the underlying structures of system call traces.

Results on a modern real-world system call datasets collected form Linux (ADFA-LD) and Windows (CANALI-LD) operating systems, have shown that the overall performances of the proposed multiple-detector ADS have significantly increased compared to the best results achieved and to those of the
best ensemble of HMMs. At an operating point of zero percent false alarm rate \((fpr = 0\%)\), the proposed multiple-detector ADS increases the true positive rate from \(tpr = 15\%\) (best result achieved) to \(tpr = 90\%\) on the ADFA-LD dataset (an increase of 500\%), and from \(tpr = 64\%\) to \(tpr = 80\%\) CANALI-WD dataset (an increase of 25\%). Our Future work involves the evaluation of combination performance of ADSs based on additional features that are complementary to system calls.

**Acknowledgment**

This research is partly supported by a grant from Natural Sciences and Engineering Research Council of Canada (NSERC), Defence Research and Development Canada (DRDC) Valcartier (QC), and Ericsson Canada.

**References**


