An Anomaly Detection System based on Variable N-gram Features and One-Class SVM

Wael Khreich[a,*], Babak Khosravifar[a], Abdelwahab Hamou-Lhadj[a], Chamseddine Talhi[b]

[a] *Software Behaviour Analysis (SBA) Research Lab, Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada*
[b] *Department of Software Engineering and Information Technology, École de technologie supérieure, Montreal, Canada*

Abstract

Context: Run-time detection of system anomalies at the host level remains a challenging task. Existing techniques suffer from high rates of false alarms, hindering large-scale deployment of anomaly detection techniques in commercial settings.
Objective: To reduce the false alarm rate, we present a new anomaly detection system based on a novel feature extraction technique, which combines the frequency with the temporal information from system call traces, and on one-class support vector machine (OC-SVM) detector.
Method: The proposed feature extraction approach starts by segmenting the system call traces into multiple n-grams of variable length and mapping them to fixed-size sparse feature vectors, which are then used to train OC-SVM detectors.
Results: The results achieved on a real-world system call dataset show that our feature vectors with up to 6-grams outperform the term vector models (using the most common weighting schemes) proposed in related work. More importantly, our anomaly detection system using OC-SVM with a Gaussian kernel, trained on our feature vectors, achieves a higher-level of detection accuracy (with a lower false alarm rate) than that achieved by Markovian and n-gram based models as well as by the state-of-the-art anomaly detection techniques.
Conclusion: The proposed feature extraction approach from traces of events provides new and general data representations that are suitable for training standard one-class machine learning algorithms, while preserving the temporal dependencies among these events.

Keywords: Software Security, Anomaly Detection Systems, Intrusion Detection and Prevention, Feature Extraction, Tracing, System calls

*Corresponding author
Email addresses: wkhreich@ece.concordia.ca (Wael Khreich), b_khosr@ece.concordia.ca (Babak Khosravifar), 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 [1, 2]. Host-based IDSs, the focus of this paper, are designed to monitor the host system activities, while network-based IDSs monitor network traffic for multiple hosts. According to their detection techniques, IDSs can also be classified into misuse detection or anomaly detection depending on whether the intrusion patterns are known or not during the design phase [3, 4]. Misuse detection techniques look for predefined patterns or signatures corresponding to known attacks, and hence they are able to achieve a high level of detection accuracy. However, misuse detection techniques cannot detect unknown attacks for which signatures have not been extracted yet (zero-day attacks) or known attacks, which are able to change their signatures with every execution (polymorphic attacks).

Typically, anomaly detection techniques construct profiles of expected *normal* behavior using training datasets that are collected over a period of normal system activity. These datasets are collected in a *secured* environment, analyzed and sanitized to ensure that the anomaly detector is trained on attack-free data [5]. During operation, the anomaly detection system 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 as they may be caused by software defects (e.g., coding or configuration errors) [6, 7]. Anomaly detection techniques are capable of detecting novel attacks, however they are prone to generate a large number of false alarms due mainly to the difficulty in obtaining a representative description of normal behavior of the system [1, 8, 9, 10]. The anomaly detectors will therefore generate an excessive number of false alarms (by misclassifying rare normal events as anomalous), which could undermine the credibility of the anomaly detection system, especially that the base-rate of normal events dominate the anomalous ones [11].

Host-based anomaly detection systems typically monitor for significant deviations in operating system calls, as they provide a gateway between user and kernel modes. Studies (e.g., [8, 9]) showed that the temporal order of system calls issued by a process to request kernel services is effective in describing normal process behavior. This has led to a considerable amount of research that investigated various techniques for detecting anomalies at the system call level (see survey in [10]). Among these, sequence time-delay embedding (STIDE) and Hidden Markov Models (HMMs) are the most commonly used [9]. However, these approaches suffer from limitations including the lack of generalization and the need for large storage capacity in the case of STIDE. Training and tuning HMM-based techniques, on the other hand, is a time and resource intensive task. The time and memory complexity of standard techniques for training HMMs grows linearly with the length and number of training sequences, and quadratically with the number of hidden states [12]. In addition, the training

process must be repeated several times for each number of hidden states, to avoid convergence to a local minimum [12].

Standard one-class machine learning techniques such as K-Nearest Neighbor (KNN), Principal Component Analysis (PCA), and One-Class Support Vector Machines (OC-SVM) have also been proposed for detecting system call anomalies. These techniques (also called semi-supervised anomaly detection since they do not require labeled data from the anomalous class) are widely available, relatively easy to apply, and provide high generalization level [13]. However, they require fixed-size feature vectors as inputs for training. The *term vector* or *bag of system calls* is the most commonly used mapping (adopted from the field of text mining and information retrieval) to transform a trace of system calls into a feature vector of binary flags [14, 15, 16, 17, 18]. Each term in this vector is typically weighted by its frequency of occurrence (called term frequency weighting) or by the term frequency–inverse document frequency [19]. As further discussed in Section 2, the term vector adopted as feature mapping in closely related works [14, 15, 16, 17, 18] ignores the temporal order of system calls within a trace, which has a negative impact on the detection accuracy.

In this paper, we propose a new anomaly detection system (ADS) that is based on one-class support vector machine (OC-SVM) trained on novel fixed-size feature vectors extracted from system call traces (and hence suitable for standard one-class machine learning algorithms), while preserving the sequential nature of system calls. The key idea is to slide a window of N system calls over the trace, associate each window with a feature vector, and compute the frequency of occurrence of each n-gram for different n values ($n = 1, 2, \ldots, N$). An n-gram is a contiguous sequence of n items from a given sequence of observation symbols. The items can, for instance, be phonemes in a sequence of speech or words in a sequence of text. In our case, an n-gram is a contiguous sequence of n system calls extracted from a system call trace. These variable length n-grams are then organized into fixed-size vectors and assigned weights corresponding to their cumulative frequency of occurrence.

These feature vectors are then used to train OC-SVM with various kernels. The resulting features provide a general mapping from traces to fixed-size feature vectors, and thus could be used to train any one-class machine learning algorithm.

We evaluate our feature extraction approach and our anomaly detection system using a modern system call dataset, called ADFA-LD[1] (Australian Defence Force Academy Linux Dataset), which has been recently made publicly available on the website of the University of New South Wales [20]. The results show that the proposed feature vectors with 6-grams provide a higher level of detection accuracy than that obtained by the term vector using both weighting schemes, the term frequency and the term frequency–inverse document frequency. More importantly, our anomaly detection using OC-SVM trained on our 6-grams feature vector achieves higher detection accuracy and lower rate of false alarms

[1] http://www.cybersecurity.unsw.adfa.edu.au/ADFA IDS Datasets/

than that of STIDE, HMM and the traditional n-grams models, and the ADS proposed by the creators of the ADFA-LD dataset [20].

The next section reviews anomaly detection systems using system call traces with a focus on the application of machine learning techniques. In Section 3, our feature design approach for mapping system call traces into feature vectors based on variable length n-grams is presented. Section 4 describes the dataset, the experimental protocol, and the evaluation metrics used in our experiments. The results are provided and discussed in Section 5. In Section 6, we discuss adversarial attacks against system call anomaly detection techniques, followed by the conclusion and future work in Section 7.

2. Sequential vs. Traditional Machine Learning Techniques for ADSs

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 [9, 10]. Normal system call traces are collected from various privileged (UNIX) processes in a secured environment using strace[2] package. Other tools for collecting system calls traces have been also used including auditd[3] and LTTng[4]. Testing system call traces are typically collected from the same processes, however when they are under attack.

The work of Forrest et al. confirmed that short sequences of system calls can describe the normal process operation, while unusual burst will occur during an attack [9]. Their anomaly detection system STIDE works by segmenting and enumerating normal (or anomaly-free) system call traces generated by a privileged process of interest into fixed-length continuous sequences, using a fixed-size sliding window, shifted by one symbol [9]. These correlations were stored in a database of normal patterns. During operations, the same sliding window scheme is used to scan the system calls generated by the monitored process for anomalies, i.e., those sequences that are not found in the normal data. These anomalies were accumulated over the entire trace (or over a temporally local region), and an alarm was raised if the anomaly count exceeded a user-defined threshold.

In a recent review, Forrest et al. [10] present and discuss various statistical and sequential machine learning and data mining techniques that have been proposed for detecting system call anomalies over the last two decades. These include rule-based techniques such as the repeated incremental pruning to produce error reduction (RIPPER) [21], which has been used for analyzing sequences of system calls and extracting rules [22, 23]. Other applications of sequential machine learning techniques include finite state automata (FSA) proposed to model the system calls language, using deterministic or nondeterministic automata [24, 25] or a call graph representation [26]. A large number of

[2] http://linux.die.net/man/1/strace
[3] http://manpages.ubuntu.com/manpages/precise/man8/auditd.8.html
[4] https://lttng.org

anomaly detection approaches is based on sequential learning algorithms including Bayesian models [27] and variations of Markovian models, such as Markov models [28, 29], variable length n-grams [30, 31], and HMMs [9, 32, 33].

Among these, HMMs have shown to provide a high level of anomaly detection accuracy, but require a large amount of training time. In fact, an HMM is a stochastic process for sequential data, determined by two interrelated mechanisms – a latent Markov chain having a finite number of states Q, and a set of observation probability distributions, each one associated with a state. Training an HMM consists of maximizing the likelihood of the training data over HMM parameters space [34]. During operation, a new (test)trace is input to the trained HMM to compute its likelihood value, which provides a degree of normality. The time and memory complexity for training an HMM with Q states (according to Baum-Welch algorithm) is $\mathcal{O}(Q^2L)$ and $\mathcal{O}(QL)$ respectively, for a trace T_{train} of length L system calls. Further details on HMMs can be found [12].

Unlike algorithms for sequential data, which can directly learn from data streams, standard one-class machine learning algorithms (e.g., KNN, PCA and OC-SVM) 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 (in fields such as text data mining and information retrieval) 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*). The *tf.idf* is a more powerful alternative weighting schema to the *tf* that provides higher weights to less frequent terms in the collection of documents than to frequent ones [19, 35]. A more formal description of *tf* and *tf.idf* is given in Section 3.

In host-based anomaly detection systems, the term vector or the *bag of system calls* (since each system call represents a term or a word and the trace represents the documents) has been proposed for detecting system call anomalies based on both, *tf* and *tf.idf*, weighting approaches [14, 15, 16, 17, 18, 36]. For instance, Liao et al. proposed an anomaly detector based on K-Nearest Neighbor (KNN) classifier using the term vector as a feature representation weighted by the *tf* and *tf.idf*, and the cosine distance as a similarity measure to discriminate normal from anomalous events [14, 37]. Rawat et al. focused on the similarity measures and proposed an alternative to the cosine measure, called binary weighted cosine (bwc), which in addition to the frequency of the system calls used by the cosine measure, considers the number of shared system calls between two feature vectors [17]. Using KNN classifiers, the authors showed that bwc can reduce the false alarms compared to cosine similarity measure. Further reduction in the false alarm rate has been shown by using extended versions of cosine and bwc measures based of radial basis kernel [18]. Kang et

al. used the term vector weighted only by the term frequency 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 [38]. 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 that of ANN and that the results are improved when the term vector is weighted by the *tf.idf* instead of just by *tf* [16].

As shown above, the term vector data representation applied to host-based anomaly detection system traditionally relies on system call frequency or their *tf.idf*, and hence discards the temporal order among the system calls in a trace. In addition, most previous work focussed on improving the distance measures between the term vectors or on experimenting with different machine learning algorithms. We believe that the sequential nature of system calls is an important characteristic that must be considered to further improve the detection accuracy and reduce the false alarm rates of ADSs based on standard one-class machine learning approaches. Therefore, we focus on designing efficient and powerful feature vectors that combine the frequency based information with the temporal information extracted from system call traces, as detailed in the next section.

Recently, Creech and Hu [39] proposed a "semantic" feature extraction technique for system call traces. The basic idea is to create feature vectors by combining all sequences of different sizes (found in the training set), up to a user defined size, such that they can account for sequences that did not 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 is not clear how the authors have trained the ELM technique using the normal traces only. They showed that their proposed approach yielded the best anomaly detection accuracy on the ADFA-LD dataset. We have compared the results of our proposed system with their results as shown in Section 5.

3. Feature Vectors based on Variable Length N-grams

This section starts by providing a formal description of both term vector weighting strategies (considered in previous work [14, 15, 16, 17, 18]). The proposed approach for efficient extraction of feature vectors based on variable length n-grams is described next.

Let $T = o_1, o_2, \ldots, o_L$ be a trace of system call observations (o_i) of length L, generated by a process with an alphabet Σ of size $m = |\Sigma|$ (unique) system calls. The collection of K traces that are generated by the process (or system) of interest and then provided for designing the anomaly detection system is denoted by $\mathcal{T} = \{T_1, \ldots, T_K\}$.

The binary term vector, $\phi(T)$, maps each trace $T \in \mathcal{T}$ into a vector of size m system calls, $T \rightarrow \phi(T)_{o \in \Sigma}$, where each term or system call $o_i \in \Sigma$ in the vector is assigned a binary flag depending on its appearance (one) or not (zero) in the trace T. The term vector can be weighted by the term frequency (tf):

$$\phi_{tf}(o, T) = freq(o_i);\ i = 1, \ldots, m \tag{1}$$

where $freq$ is the number of times system call o_i appears in T, normalized by L (the total number of system calls in T).

The term frequency considers all terms as equally important across all documents or collection of traces ($\mathcal{T}$). However, rare terms that appear frequently in a small number of documents convey more information than those that are frequent in most documents. The inverse document frequency (idf) is proposed to increase (or decrease) the weights of terms that are rare (or common) across all documents. The term vector weighted by the *tf.idf* is therefore given by:

$$\phi_{tf.idf}(o, T, \mathcal{T}) = \frac{K}{df(o_i)} freq(o_i);\ i = 1, \ldots, m \tag{2}$$

where the document frequency $df(o_i)$ is the number of traces T_k in the collection $\mathcal{T}$ of size K that contains system call o_i. A high weight in *tf.idf* is thereby given to system calls that are frequent in a particular trace $T \in \mathcal{T}$, but appear in few or no other traces of the collection $\mathcal{T}$. There are several variants to Equation 2, which, for instance, take the logarithm of inverse document frequency or apply other normalization factors [35]. However, as shown in Equation 1 and 2, both weighting strategies discard the temporal order of system calls.

The proposed approach however accounts for the temporal order of system calls by extracting and mapping variable length n-grams and their frequencies from each trace $T \in \mathcal{T}$ to fixed-size feature vectors. Each n-gram is a sequence of contiguous system calls of length n extracted from trace T.

As illustrated in Figure 1, the feature extraction starts by sliding a window of N system calls over the trace $T \in \mathcal{T}$, shifted by one system call. For each sequence, the individual (or 1-gram) system calls are first extracted, followed by all n-grams for $n = 1, \ldots, N$ that are rooted at the first system call inside the sliding window, which are then organized in vectors V_i (see Figure 1). The size of each vector V_i is at maximum $N + N - 1 = 2N - 1$ n-grams, when no duplicate system calls or n-grams appear inside the window. Otherwise, if the same system call is repeated within the sliding window, the size of V_i will have smaller size. For a trace of length L and a sliding window of length N, the total number of vectors is $L - N + 1$, however the number of unique vectors could be lower in practise depending on the regularity of the process generating the data.

The unique n-grams (for $n = 1, \ldots, N$) from all unique vectors V_i obtained by sliding the window over the available traces $T \in \mathcal{T}$ are used as dictionary keys, while their accumulated frequencies are used as values. This dictionary of size, say D, is therefore the reference database representing the normal process behavior by the frequencies of the 1-grams, 2-grams, ..., N-grams that occurred

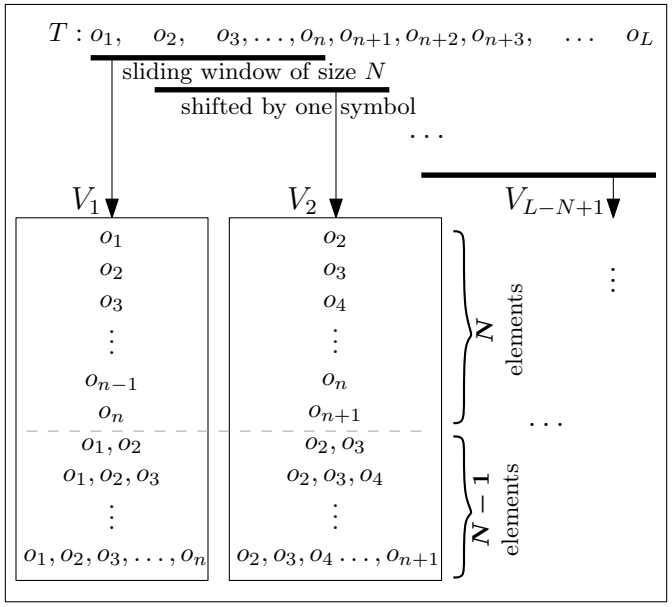


Figure 1: An illustration of n-grams vector extraction from a trace T of L system calls, using a sliding window of size N.

in the collection of traces $\mathcal{T}$. Finally, each vector V_i (shown in Figure 1) is mapped to the space D of the reference dictionary and becomes a feature vector. The only non-null elements of the feature vector are those that correspond to the n-grams of the original vector (V_i) before the mapping. Those elements are weighted by their frequencies that have been accumulated as values in the reference dictionary.

For a specific process, the size of the dictionary (D), which is the size of the feature vectors, depends on the alphabet size m, the sliding window size N and the regularity of the process. In practise, the dictionary size of even complex processes is manageable as further discussed in Section 4. For instance, the ADFA-LD dataset has an alphabet of size $m = 175$ system calls. Therefore, the obtained dictionary sizes are $D = 12,830$ for $N = 3$ and $D = 142,190$ for $N = 6$. However, the obtained feature vectors are very sparse, at most $2 \times 6 - 1 = 11$ elements are non-null, and hence they can be efficiently encoded in practise using specialized machine learning libraries for sparse vectors and matrices computation [41].

As described above, the feature vectors comprise the n-grams extracted from the sliding window of size N, weighted by their frequencies of occurrences in the trace. The value of N is a user-defined parameter that influences the detection power and the size of the feature vectors. A small N value is always desirable since it results in smaller feature vectors, and hence allows faster detection and response during operation. Moreover, it can effectively distinguish the anomalies from the normal sequences of the attack trace. In contrast, very

small N could result in misclassification of anomalies because the system cannot extract information about temporal order of system calls in the trace. Note that taking $N = 1$ reduces the feature vector to the term frequency, which only considers the frequency of individual system calls without any temporal information. On the other hand, larger N values incorporate more temporal information into the feature vectors and hence expected to improve the detection accuracy, but increase the time and space required for extracting the feature vectors (during the training phase).

In practise, choosing the smallest N values that provide good detection accuracy depends on the size of the smallest anomalous sequence in the testing set. Some evasion attacks (see Section 6 for a detailed discussion) rely 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 [42]. An example of such *blind regions* is provided by Tan et. al 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 [42]. 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. In our experiments, we provide and compare the results for $N = 3$ and 6 system calls as detailed in the following section.

4. Experiments

The objectives of the experiments are to compare the performance of our variable length n-gram feature vectors (for different N values) to that of the term vector, weighted by *tf* and *tf.idf*, as commonly proposed in related work. More importantly, we evaluate and compare the performance of our anomaly detector to those based on the most commonly used sequential modeling techniques, such as STIDE, HMMs and traditional n-grams detectors, as well as to the ADS proposed by Creech and Hu [39]. The performance evaluation is conducted on ADFA-LD dataset (described next) according to the experimental protocol described in Section 4.2, and assessed using the Receiver Operating Characteristics (ROC) curves and Under the Curves (AUC) as further described in Section 4.2.4.

4.1. ADFA-LD Dataset

The generation of system call datasets for designing and evaluating host-based ADSs is typically performed in two phases. Normal system call traces are first collected during normal operation of the process or system in a secured environment. These traces are assumed attack-free and used for training the anomaly detectors. The testing traces are generated by collecting the system calls from the same host while being under attacks. These attack traces comprise both normal and anomalous sequences for testing, however it is difficult to isolate the manifestation of an attack within the trace. Therefore, during testing, the whole attack trace or the collection of attack traces is considered as one anomaly.

Table 1: Summary of ADFA-LD system call dataset

	Number of traces	Number of system calls
Training data (normal)	833	308,077
Testing data (normal)	4,373	2,122,085
Attack data (anomalous)	746	317,388
Total	5,952	2,747,550

The ADFA-LD has been recently created and made publicly available on the website of the University of New South Wales (UNSW) [20]. It is currently the most recent and representative dataset available for benchmarking of the anomaly detection techniques based on system call sequences. The UNM dataset has been the most commonly used system call dataset [8]; however, it is now over 20 years old, and hence both the normal and the attack traces are no longer representative to the complexity of the current systems and to sophistication of the modern attacks.

The ADFA-LD dataset is generated using a modern operating system and servers attacked by exploiting various (publicly known) security vulnerabilities. As described by the authors, the ADFA-LD is generated using a fully patched Ubuntu Linux 11.04 operating system with an Apache 2.2.17 web server, PHP 5.3.5 server side scripting engine, TikiWiki 8.1 content management system, FTP server, MySQL 14.14 database management system and an SSH server [20]. Normal system call traces were generated from the host system during normal user activities, such as web browsing and Latex document preparation. These traces were collected using Linux audit daemon (auditd[5]), an auditing framework for collecting and tracking security audit trails. As described in Table 1, the ADFA-LD dataset comprises 833 normal traces for training and 4373 normal traces for testing.

The dataset comprises 746 traces generated from 60 different attacks, belonging to six types of attack vectors [20]. These attacks were launched by a certified penetration tester against the system, using modern penetration testing tools like Metasploit[6] framework. The attack framework includes client and server side attacks as well as social engineering techniques. The client side attacks focused on initiating connections from the target machine, for example by a exploiting poisoned executable or Trojaned programs. The server side attacks relied on classical hacking methods, where open services on the target are used to gain a privileged command shell. The attack vectors include remote password brute force attacks using the parallelized login cracker Hydra[7] over FTP and SSH. Client side poisoned executables (e.g., malicious executables with backdoors, or Trojan horses) have been used to add new superuser

[5] http://manpages.ubuntu.com/manpages/precise/man8/auditd.8.html
[6] http://www.metasploit.com
[7] http://www.thc.org/thc-hydra/

Table 2: Summary of ADFA-LD attack structure and distribution.

Attack	Payload/Efect	Vector	#Attacks	#Traces
Hydra-FTP	Password bruteforce	FTP by Hydra	10	162
Hydra-SSH	Password bruteforce	SSH by Hydra	10	176
Adduser	Add new superuser	Client side poisoned executable	10	91
Java-Meterpreter	Java-based Meterpreter	TikiWiki vulnerability exploit	10	124
Meterpreter	Linux Meterpreter payload	Client side poisoned executable	10	75
Webshell	C100 Webshell	PHP remote file inclusion vulnerability	10	118

accounts to hijack the system, using the Linux Meterpreter payload. The Meterpreter is an enhanced command shell provided by the Metasploit framework, which simplifies the compromise phase, and is available in several formats, including Java and standalone executables. The TikiWiki vulnerability is another vector of attack used to upload a copy of connection to the attacking system by using the Meterpreter, an enhanced functionality command shell provided by the Metasploit framework.

Simulated social engineering methods are also used to directly manipulate the user of the target machine to access malicious payloads and hence open an access point for an attacker. A PHP remote file inclusion vulnerability is achieved using a C100 web shell (a PHP code which provides a graphical interface to the attacker) as the last payload, leveraging the remote file inclusion vulnerability of the TikiWiki installation[8]. The TikiWiki vulnerability is also used to upload a copy of the Java Meterpreter payload, which initiated a reverse TCP connection to the attacking computer when executed. Once the shell was established, several attempts were conducted to privilege escalation, access the shadow password file, and install backdoor tools. Table 2 describes the details and presents the distribution of each attack. For further details regarding the dataset, normal and attack data generation, the reader is referred to the original paper by Creech and Hu [20].

4.2. Experimental Protocol

This section provides details about feature extraction, model training and evaluation metrics employed in our experiments. It starts by describing the detection approaches, which are based on computing the Euclidean distances between the feature vectors or on training support vector machines using those vectors. Training of STIDE, HMMs and the traditional n-grams models as a reference sequential techniques for comparison is described next, followed by a description of the performance evaluation metrics used in the results section.

We followed the experimental setup provided in [39]; therefore, the 833 normal traces are used for training STIDE, HMM, and OCSVM detectors. However, we held 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 tune models parameters, such as the number of nearest neighbors

[8] http://www.exploit-db.com/exploits/18265/

(K), the Window Size (W) for STIDE detector, the number of states for HMMs and the kernel parameters for OCSVM. The remaining (3373) normal traces and (40) attack traces, in Table 1, 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].

First, we applied the term vector weighted by the *tf* and *tf.idf* as well as our variable length n-gram feature vectors (using $N = 3$ and $N = 6$) to both normal and test traces from the ADFA-LD dataset (Table 1). The frequency of each term in the trace is used as a weight for the *tf* vector (see Equation 1). For the *tf.idf*, the term vectors are weighted according to Equation 2, where a document is the trace (or traces) belonging to one process, which is defined by its PID. The term vector (using *tf* or *tf.idf* weighting schemes) provides one feature vector for each trace.

We then extracted two sets of variable length n-gram feature vectors, using a sliding window of size $N = 3$ and $N = 6$. We denote by $VN3$ the sets of feature vectors for variable n-grams up to size 3 and by $VN6$ the sets for variable n-grams up to size 6. In contrast to the term vector, our approach extracts $L - N + 1$ vectors are provided with our variable length n-gram features (for a trace of size L), as described in Section 3.

To evaluate the detection accuracy of each feature extraction technique, we used two anomaly detection approaches: a distance-based approach and a more complex detector based on one-class support vector machine, as described next.

4.2.1. Distance-based Detection Approach

The first detection approach is based on Euclidean distance between the normal training traces and the testing traces, which may involve one or a number of feature vectors. We have chosen the Euclidean distance measure because of its simplicity and well understood meaning, such that we can investigate the discriminative power of the feature vectors more clearly. Each feature extraction technique, (i.e., *tf*, *tf.idf*, $VN3$, and $VN6$) is applied to extract the corresponding feature vectors from the normal traces belonging to the training dataset (described in Section 4.1). Hereafter, we will refer to the reference feature vectors extracted from the training set by "normal vectors". Similarly, these feature extraction techniques are also applied to extract the corresponding feature vectors from the test traces (which include both normal and attack traces). Depending on the feature mapping techniques, each testing trace can result in one vector (such as the case of *tf* and *tf.idf*) or several vectors (as for $VN3$ and $VN6$).

We then compute the Euclidean distance from each vector resulting from the test set, v_j^{test}, to each normal vector v_r^{norm} in the training set, according to Equation 3:

$$d_r(j) = \sqrt{\sum_{i=1}^{D} (v_r^{norm}(i) - v_j^{test}(i))^2} \tag{3}$$

where D is the size of the feature vector (as described in Section 3), $r = 1, \ldots, R$, and $j = 1, \ldots, J$. R represents the number of feature vectors computed during training (for the normal vectors) while J is the number of feature vectors extracted from the test set. Once the Euclidean distances from each test vector v_j^{test} to each normal vector v_r^{norm} are computed, they are sorted in increasing order. The average of the closest K distances to normal vectors (v_R^{norm}) is then considered as the score, which is similar to K-Nearest Neighbors (K-NN):

$$d_{avg}(j) = avg(min(d_k(j))),\ k = 1, \ldots, K \leq R \tag{4}$$

When $K = 1$, that means only the minimum distance to the normal vectors is considered as a score, which is similar to the Nearest Neighbor (1-NN):

$$d_{min}(j) = min(d_k(j)),\ k = 1, \ldots, K \leq R \tag{5}$$

For the term vector models (*tf* and *tf.idf*), which generate one feature vector for each trace. However, as described in Section 3, the proposed feature extraction approach produces $L - N + 1$ feature vectors for each trace of length T, using n-grams up to length N. Therefore, the average distances of these vectors is considered when computing the distances from each test trace to the normal vectors in Equations 4 and 5. Finally, the resulting distance measures are used as scores to generate the ROC curves and compute the AUC values as described in Section 4.2.4.

4.2.2. OC-SVM based Detection Approach

One-Class Support Vector Machine (OC-SVM) is a powerful and commonly used machine learning method in various domains [43, 44]. Essentially, the OC-SVM algorithm maps the input data into a high dimensional feature space using kernel functions and then attempts to find the maximal margin hyperplane that separates the training data points from the origin [43]. This formulation is similar to that of the two-class SVM, however the OC-SVM considers that the normal training data are far from the origin, while the anomalous data lie in the neighborhood of the origin. There is an equivalent formulation that uses a hypersphere to describe the data in the feature space, and tries to find the smallest hypersphere that contains most of the data [44].

The hyperplane that separates normal from anomalous data corresponds to the classification rule [43]:

$$f(v) = w^T \cdot v + b \tag{6}$$

which represents the dot product of the normal vector (w) and a bias term (b). The optimization problem consists therefore of finding the rule f with a maximal geometric margin. This classification rule can be used to classify a test input v^{test} as anomalous if $f(v^{test}) < 0$; or normal otherwise. In practice, there is always a trade-off between maximizing the distances of the hyperplane from the origin and the number of normal data points contained in the other region separated by the hyperplane. The distance from the hyperplane to a test example v^{test} could also be used as score or degree of membership to the normal or anomalous class; these scores are then used to generate the ROC curves.

Our proposed anomaly detection system is based on OC-SVMs detectors trained using our sets of feature vectors ($VN3$, and $VN6$) extracted from the (normal) training traces. Training OC-SVMs is done using LIBSVM[9], a library for support vector machines. In our experiments, we have trained and compared the performance of OC-SVMs using two commonly used kernels: linear kernel and Gaussian kernel.

The linear kernel $K(v_i, v_j)$ is simply the dot product of the two given vectors, and can be computed according to Equation 7.

$$K(v_i, v_j) = v_i.v_j \tag{7}$$

The Gaussian (or RBF) kernel can be computed according to Equation 8:

$$K(v_i, v_j) = e^{\frac{-||v_i - v_j||^2}{2\sigma^2}} \tag{8}$$

where σ^2 denotes the variance.

For comparison, we also trained the OC-SVMs using the term vectors (with *tf* and *tf.idf* weights). The results are presented and discussed in Section 5.

4.2.3. Sequential Models

As described previously, the sequential models considered in this work are STIDE [9, 10], HMMs [32, 33], and the traditional n-gram models [28, 29], since they have been shown to provide a high level of detection accuracy. Building the normal database for STIDE only requires the selection of the sliding window size (W). In our experiments, we trained several window sizes $W = \{6, 10, 20\}$, and reported the results for $W = 6$, since it provided the best false and true positive performance on the held out validation dataset. Training, the traditional n-grams model consists of computing the probability of each system call in the subsequence of n system calls conditioned on the $n-1$ previous system calls. The time and resource requirements for training n-gram grow exponentially with n and could be prohibitive for large alphabet set. In our experiments, we selected $n = 6$ for a fair comparison of performance with STIDE and our $VN6$ feature vectors, which contains up to 6-grams.

As described in Section 2, estimating the parameters of an HMM requires the specification of the number of output symbols (M) and, more importantly, the number of hidden states Q. The number of output observation symbols is taken equal to the host system alphabet size, i.e, $M = 175$ unique system calls for the ADFA-LD dataset, as shown in Table 1. Since using a single HMM with a pre-specified number of states may have limited capabilities to capture the underlying structure of the data [32, 12], 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 [34]. To reduce overfitting effects, the evaluation of the log-likelihood on the independent

[9] `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

validation set (that contains a 1000 of the normal traces, which are not used during training nor testing) 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. In Section 5, we report the results of HMM with $Q = 200$ states since it provided the best log-likelihood value on average.

The Forward-Backward algorithm is then used to evaluate the performance of the trained HMM on the test set, which should assign significantly lower log-likelihood values to anomalous sequences than to the normal ones [12]. The log-likelihood values provided by HMM to each test traces are used to generate the ROC curves and compute the AUC values, as described next.

4.2.4. Evaluation Measures

In general, the anomaly detector provides scores or probabilities of membership to the normal class for each subsequence of the testing set (extracted from the trace using the sliding window technique). The lower the score the higher the likelihood of that subsequence being anomalous. In our work, the impact on performance of using the different features, models, and detection approaches is assessed using the ROC analysis [45].

A ROC curve is a plot of true positive rate (tpr) against the false positive rate (fpr) for all decision thresholds as shown in Figure 2. The tpr is the proportion of attack traces correctly detected over the total number of attacks in the test set. The fpr is the proportion of normal traces incorrectly classified as anomalous over the total number of normal traces in the test set. A crisp detector (i.e., STIDE) produces a single data point in the ROC plane since it directly produces a class label (i.e., normal or anomaly). In contrast, a soft detector (e.g., HMM and OCSVM) assigns scores or probabilities to the input samples, which can be converted to a crisp detector by setting a threshold on the scores. A soft detector produces a ROC curve by varying the decision thresholds over the complete range of scores. ROC curves can be efficiently generated by sorting the output scores provided by the detection models from the most likely to the least likely value, and considering unique values as decision thresholds [45].

As illustrated in Figure 2, the ROC curves allows to visualize the performance of detectors and select optimal operational points, without committing to a single decision threshold. It presents detectors' performance across the entire range of class distribution and error costs, but it is more important to analyse the shape of the curves at the regions of interest (i.e., at low fpr values in anomaly detection). For equal prior probability and cost of errors, the optimal decision threshold (minimizing overall errors and costs) corresponds to the vertex that is closest to the upper-left corner of the ROC plane. In many practical cases, where prior knowledge of skewed class distributions or misclassification costs are available from the application domain, they should be considered while analyzing the ROC curve and selecting operational thresholds, using iso-performance lines [45].

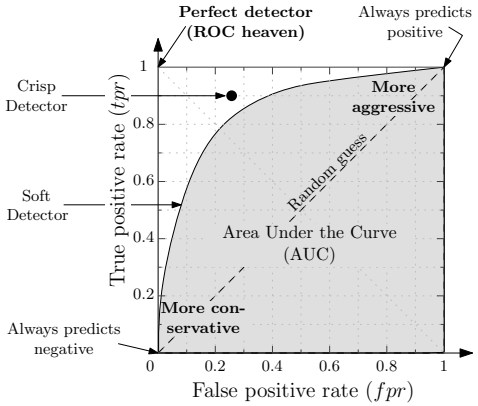


Figure 2: Illustration of the performance of a crisp detector (single point) and a soft detector (ROC curve generated for all decision thresholds). Important regions in the ROC space are annotated.

The area under the ROC curve (AUC), has been proposed as more robust scalar summary of detectors' performance than accuracy (acc) [46]. The AUC provides a global measure for comparing detectors' performance independently of the decision thresholds; it could be interpreted as the average of the tpr over all values of the fpr. An $AUC = 1$ indicates a perfect detector, which detects all anomalies without any false alarms ($tpr = 1, fpr = 0$), while a random detector will have an $AUC = 0.5$. For a crisp classifier, the AUC is simply the area under the trapezoid and is calculated as the average of the tpr and fpr values. For a soft classifier, the AUC may be estimated directly from the data either by summing up the areas of the underlying trapezoids [45] or by computing of the Wilcoxon-Mann-Whitney (WMW) statistic [47].

However, when the ROC curves cross, detectors providing higher overall AUC values may perform worse than those providing lower AUC values in a specific region of ROC space, which could be the region of interest for the application. In such case, the partial area under the ROC curve (pAUC) could be more useful for comparing performances in the specific regions of interest [48]. Therefore, in our experiments we provide the results in terms of ROC curves, which allows to visualize the performance of each detector across the entire region. Furthermore, we provide the partial area under the ROC curve, pAUC, for the range of $fpr = [0, 0.1]$. In addition, we present the tpr and the accuracy values at two fixed operating points: $fpr = 10\%$ and $fpr = 5\%$.

5. Results and Discussions

We first present the results of the distance-based detection approach applied to each feature extraction technique (*tf*, *tf.idf*, $VN3$, and $VN6$). The objective is to investigate the discrimination power of each feature independently from the model, since the distance based approach is parameter free. We then provide

the results of our ADS based on the OC-SVM and trained using our feature vectors, compared to those of the sequential models.

5.1. Comparison of Feature Vectors using the Distance-based Approach

Figure 3 illustrates the ROC curves and pAUC values of the distance-based detection approach applied to the features vectors, *tf*, *tf.idf*, $VN3$, and $VN6$ extracted from ADFA-LD dataset. Figure 3a presents the ROC curves of the minimum distance (1-NN), computed according to Equation 5, while those in Figure 3b are computed for $K = 10$ according to Equation 5according to Equation 4.

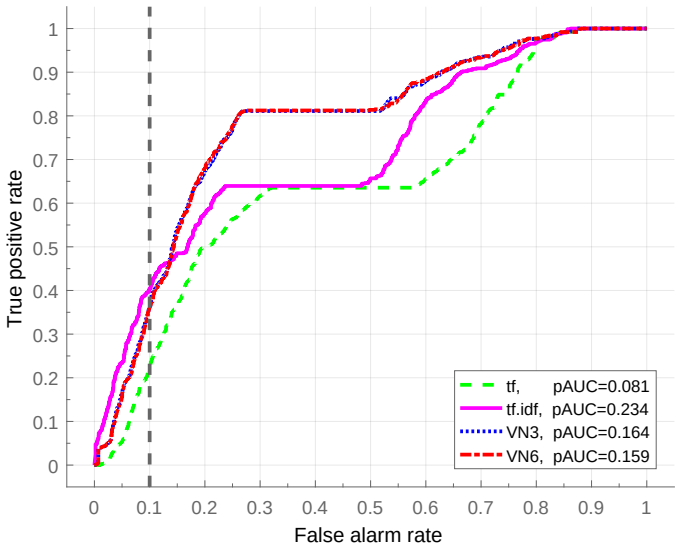
As described in Section 4.2.4 the closest ROC curve to the upper-left corner of the ROC plane, the better the performance, since such curve provides lower false positive rate and higher true positive rate. The partial AUC values, shown in the legend, are computed for the range of $fpr = [0, 0.1]$, as described in Section 4.2.4. Other metrics, such as the true positive rate and the accuracy values at a false positive rate of 10% and 5% are also presented in Table 3. The higher the value of pAUC the better, in the ideal case when the ROC curve reaches the point (0, 1) in the ROC space, the pAUC value will be one.

As shown in Figure 3, the ROC curves of our proposed feature vectors, containing sequential information up to 3-grams, $VN3$, outperform the ROC curves of *tf*, *tf.idf* feature vectors in large areas of the ROC space. In the low false positive region (where $fpr < 10\%$) all feature extraction techniques provide comparable performance as shown by the shape of the ROC curves and by the partial AUC values (pAUC in the legend). This can be also seen in accuracy values presented in Table 3.

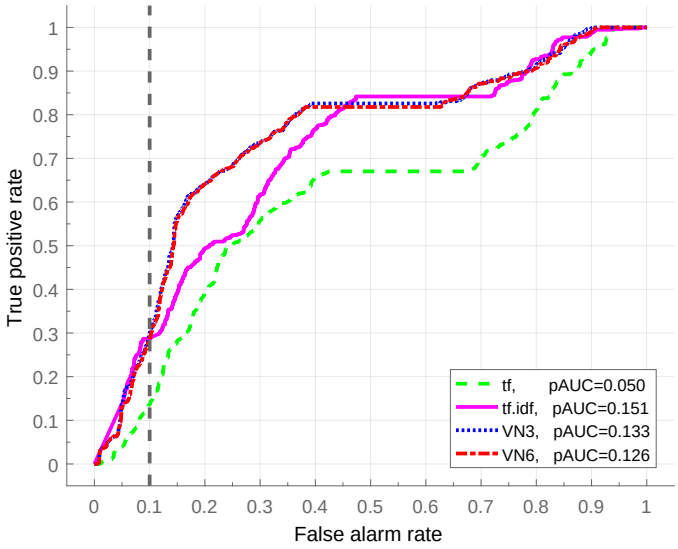
We have experimented with other K values, using the validation set, and noticed a slight decrease in the performance of the ROC curves for all feature extraction techniques with the increase of K. Overall, considering the minimum distance (K=1) provided the best results. Unexpectedly, both $VN3$ and $VN6$ features provide almost similar ROC curves in both Figures 3a and 3a, which could be attributed, to the simplicity of the distance based approach. These results show that our feature vectors $VN3$ and $VN6$, in general, outperform the traditional term vectors with both weighting schemes *tf* and *tf.idf*. For further insight into the performance of the proposed features, however, we included the term vectors techniques (*tf* and *tf.idf*) in the next set of experiments.

5.2. Results of the proposed ADS: OC-SVMs trained on our Feature Vectors

This section presents the results obtained by the proposed ADS using OC-SVMs with linear and Gaussian kernels (as presented in Section 4.2.2), trained on our feature vectors ($VN3$, and $VN6$) for detecting system call anomalies in the ADFA-LD datasets (presented in Section 4.1). As stated previously, the results of OC-SVMs trained using the term vector (with *tf* and *tf.idf* weights) are also presented for comparison. We compare our results to those achieved by STIDE, HMMs and n-grams models as well as by the ADS proposed by Creech and Hu [39], which we refer to by CH2013 hereafter.

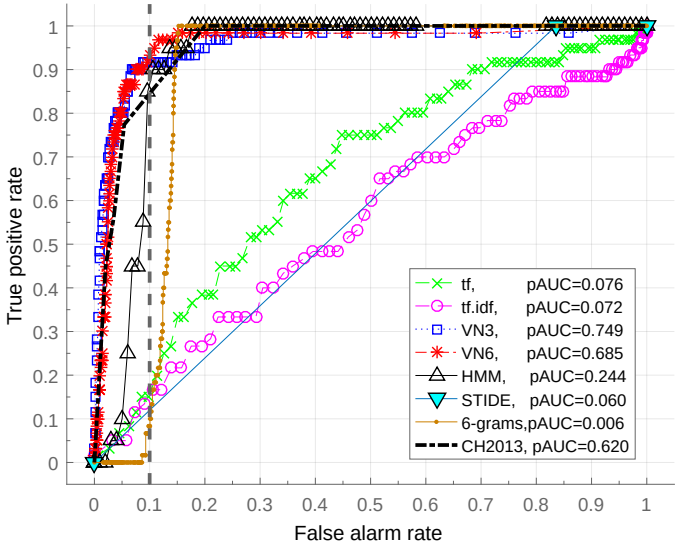


(a) ROC curves using the minimum distance to the closest neighbor (1-NN).

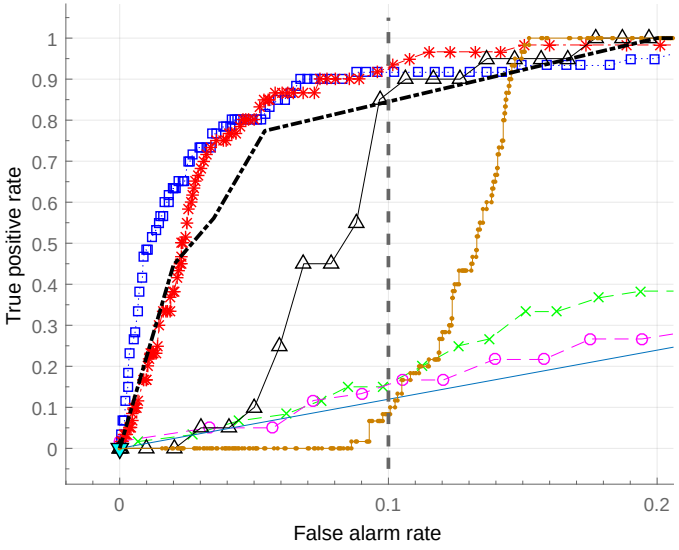


(b) ROC curves using the average distances to the 10 closest neighbors (10-NN).

Figure 3: ROC curves and pAUC values produced by the distance-based detection approach using the four sets of feature vectors extracted from ADFA-LD dataset.

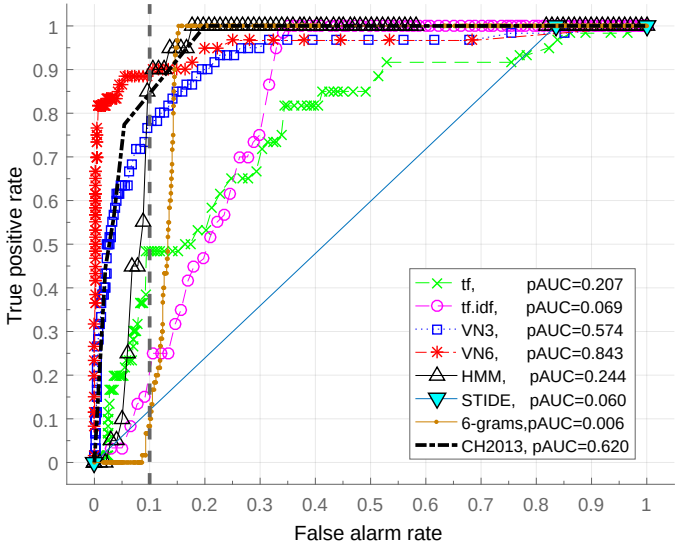


(a) The entire ROC space.

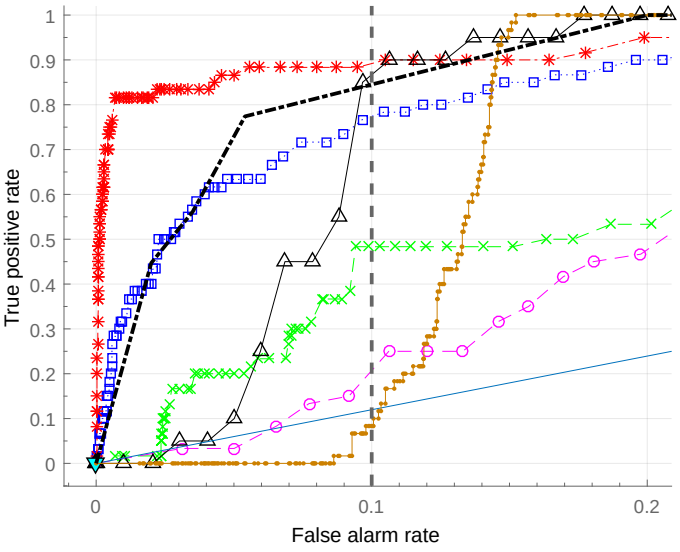


(b) ROC curves of Figure 4a with a focus on the region of interest, low fpr.

Figure 4: ROC curves and pAUC values of the OC-SVM detector using a linear kernel and trained on our feature vectors ($VN3$ and $VN6$), compared to those of STIDE and HMM and to results presented in [39] (CH2013).



(a) The entire ROC space.



(b) ROC curves of Figure 5a with a focus on the region of interest, low fpr.

Figure 5: ROC curves and pAUC values of the OC-SVM detector using a Gaussian kernel and trained on our feature vectors ($VN3$ and $VN6$), compared to those of STIDE and HMM and to results presented in [39] (CH2013).

The ROC curves and pAUC values are illustrated in Figure 4 and 5. As shown in the curves presented in Figure 4a and more clearly in the zoomed region of Figure 4a, the ROC curves of the proposed ADS using OC-SVM trained on our $VN3$ and $VN6$ feature vectors dominate all other curves. The OC-SVM using a linear kernel achieves the overall best pAUC results for both feature vectors $VN3$ and $VN6$, as illustrated in the legend of Figure 4. The ROC curves and pAUC values of OC-SVMs trained on $VN3$ and $VN6$ outperform that of HMM and CH2013. This shows that even a linear kernel was able to better separate the normal and attack traces in the high-dimensional space provided by the proposed feature vectors, led to improved generalization capabilities, and hence an increase in the detection rates while reducing the false alarm rates.

In contrast, the OC-SVMs using the same kernel but with the term vector features (*tf* and *tf.idf*) perform poorly (almost close to random detectors), as shown in Figure 4. This is an unsurprising result, since both term vectors with *tf* and *tf.idf* weights ignore the temporal order of system call. However, Figure 4 shows some signs of overfitting, since the linear kernel OCSVM trained on $VN3$ performs slightly better than that trained on $VN6$. Therefore, it may be better to consider the length of sliding window (N) as a parameter to optimize during training and validation to avoid overfitting the training data. Finally, as expected STIDE is shown to performed poorly, due to its lack of generalization and because the space of normal behavior is large and complex for a dataset with an alphabet size of 175 system calls.

The results achieved with OC-SVMs using Gaussian kernels on ADFA-LD dataset are presented in Figure 5. The ROC curves of the OC-SVM trained on our $VN6$ feature vectors dominates all other curves and achieves the overall highest pAUC value among all competing techniques. In particular, the ROC curve of OC-SVM trained on $VN6$ shows that it is able to detect 86% of the attacks at a false alarm rate of 5%, which is significantly better than all other techniques, including the sequential models. On the other hand, and similarly to the OC-SVMs trained with a linear kernel, OC-SVMs trained with a Gaussian kernel using the term vector features (*tf* and *tf.idf*) perform poorly, as shown in Figure 5.

Table 3 summaries the pAUC values, the true positive rate and the accuracy values at a false positive rate of 10% and 5% for all anomaly detection techniques. These results show that the generalization capabilities of OC-SVM combined with the temporal information included in $VN6$ feature vectors provide an anomaly detector that can outperform state-of-the-art detectors. In addition, they illustrate a large improvement in detection accuracy of both feature vectors $VN3$ and $VN6$ over that of *tf* and *tf.idf*. Finally, the results of the experiments confirm the importance of using the temporal information for detecting system call anomalies, and demonstrate a significant reduction in false alarm rate and improvement of true positive rate when our variable n-gram feature vectors, $VN6$, is used to train powerful detector with high generalization capabilities such as the OC-SVM.

Table 3: Summary of the partial area under the ROC curve (pAUC), as well as true positive rate (tpr) and accuracy (acc) values computed at two false positive rate (fpr) values for all techniques.

			$fpr = 10\%$			$fpr = 5\%$	
	Feature	AUC	pAUC	tpr	acc	tpr	acc
1-NN	*tf*	0.648	0.081	0.223	0.801	0.052	0.819
	tf.idf	0.717	0.234	0.403	0.828	0.223	0.846
	$VN3$	0.778	0.164	0.359	0.821	0.168	0.836
	$VN6$	0.778	0.159	0.361	0.821	0.155	0.834
10-NN	*tf*	0.604	0.050	0.135	0.789	0.039	0.817
	tf.idf	0.710	0.151	0.287	0.811	0.139	0.832
	$VN3$	0.746	0.133	0.300	0.813	0.137	0.831
	$VN6$	0.741	0.126	0.284	0.810	0.129	0.830
OC-SVM Linear Kernel	*tf*	0.660	0.076	0.158	0.890	0.072	0.938
	tf.idf	0.555	0.072	0.156	0.890	0.050	0.938
	$VN3$	**0.955**	0.749	0.917	**0.900**	0.800	0.948
	$VN6$	**0.955**	0.685	**0.929**	**0.900**	0.803	0.948
OC-SVM Gaussian Kernel	*tf*	0.767	0.207	0.483	0.894	0.200	0.940
	tf.idf	0.799	0.069	0.207	0.891	0.033	0.938
	$VN3$	0.917	0.574	0.774	0.898	0.633	0.946
	$VN6$	0.953	**0.843**	0.892	**0.900**	**0.867**	**0.949**
HMM	traces	0.919	0.244	0.867	**0.900**	0.099	0.938
STIDE	sequences	0.582	0.060	0.120	0.889	0.060	0.938
6-grams	sequences	0.884	0.006	0.083	0.889	0.000	0.937
CH2013	semantic	0.954	0.620	0.845	0.899	0.729	0.947

5.3. Threat to Validity

We have conducted experiments using only one system call dataset derived from the Linux operating system, which consists a threat to external validity of this study. More experiments are therefore required to generalize the presented results to other operating systems and other programs.

A threat to internal validity exists in the implementation of anomaly detection techniques, STIDE, HMMs, OC-SVMs and the feature extraction techniques, *tf*, *tf.idf*, $VN3$, and $VN6$, as well as in conducting the experiments, preprocessing, and splitting the datasets.

The selection of attacks is one of the common threats to validity for approaches aiming to detect anomalies. It is possible that the attacks in terms of execution paths do not exhibit large variation among the possible attack space. This may impact our results. However, to our knowledge, the ADFA-LD dataset is the only publicly available modern dataset for system calls.

6. Discussion of Evasion Techniques and Challenges

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 [49].

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 [26, 50]. 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 [51, 52, 53, 54], memory and call stack information [55, 56, 25], and function calls and other user-space information [57, 58]. 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.

An attacker could also attempt to predict the threshold that raises the alarm in order to make his attack go undetected (under the radar) [59]. As described in Section 4.2.4, 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 the original work of Warrender et al., since the anomaly signal is computed from the number of mismatches occurring in a temporally local region [9]. 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. In our experiments, we did not use such a smoothing threshold to the reduce false alarms; any normal sequence given a score (by our detector) below the original decision threshold is considered as a false alarm, 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 [50]. Recent works included additional information about the system call arguments to defend against such attacks [51, 52, 53, 54]. However, these approaches have difficulties in deciding which legitimate argument value is really benign, when multiple legitimate values appear in the training phase [60].

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 [56]. In contrast, the white-box detectors examine the program being monitored by statically analyzing the source code or binary [61, 62, 63, 26]. 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 [56]. Sekar et al. proposed the first gray-box anomaly detector, by including the program counter of the process with the system call number [25], while Feng et al. further incorporated the return addresses on the call stack of the process when each system call is invoked [55]. Tandon and Chan coupled the system call arguments with their return values [54]. 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 [57]. Mutz et al. further extended the previous approach by 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 [58].

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 [64]. The authors focused on demonstrating the ability of their symbolic execution technique to generate configurations that can return control back 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 [65]. The authors demonstrated exploits against data-flow vulnerabilities by, for instance, using normal system calls to overwrite the password file and then elevate privileges [65]. Bhatkar et al. applied similar kind of attacks 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 [66]. 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 [67]. 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 in practice without disrupting the order of the system calls. As shown in [68], 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.

Our ADS based on the proposed feature vectors applied to train OC-SVM using Gaussian kernel have shown a significant reduction in the false alarm rate and increase in true positive rate compared to state-of-the-art anomaly detectors based on the system all sequences. We think that the key problem of anomaly detection system, in practise, 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 ADS that rely on additional information about the system call arguments, return values, call stack, function calls, etc. 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.

However, several research issues remain open. In particular, the evaluation of the performance and interaction among several ADSs based on different features and the possibility of crafting attacks that evade detection. Adapting the anomaly detector to changes in the normal behavior over time in order to reduce the false alarm and maintain or improve the detection accuracy is another important challenge.

7. Conclusion

In this paper, we present an ADS based on OC-SVM and a new approach for designing feature vectors that combines the frequency with the temporal information extracted from system call traces. The proposed approach accounts for the temporal order of system call within a trace by extracting and mapping variable length n-grams and their frequencies to fixed-size feature vectors. Our feature vectors provide therefore new representations of the system call traces that are suitable for training standard one-class machine learning algorithms, while preserving the sequential nature of system calls. The proposed feature vectors with sequential information up to 3-grams ($VN3$) and 6-grams ($VN6$) coupled with the generalization capabilities of OC-SVM are able to increase the detection accuracy while reducing the number of false alarms.

The proof-of-concept experiments are conducted on a benchmark system call dataset obtained from the University of South Wales. The results show that the proposed OC-SVM detector with a Gaussian kernel trained on our feature vectors is able to provide higher level of detection accuracy than that achieved by previous techniques, using the term vector features with both weighting schemes (term frequency and term frequency–inverse document frequency). More importantly, the OC-SVM using the sequential information up to 6-grams ($VN6$) provided with our feature vector achieves the overall highest detection rate with the lowest false positives compared to STIDE, HMM and the results obtained by the creators of the ADFA-LD dataset.

As future work, we plan to conduct more experiments on other datasets collected from other operating systems to confirm the validity of our results. We are currently with the Defence Research and Development Canada (DRDC) Valcartier (QC) to apply these techniques to real-world settings. Moreover, we intend to investigate the combination of multiple machine learning techniques trained on the proposed feature vectors with sequential learning detectors.

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

[1] D. E. Denning, An Intrusion Detection Model, in: Proceedings of the Seventh IEEE Symposium on Security and Privacy, 1986, pp. 119–131.

[2] J. McHugh, A. Christie, J. Allen, Defending yourself: the role of intrusion detection systems, Software, IEEE 17 (5) (2000) 42–51. doi:10.1109/52.877859.

[3] S. Axelsson, Intrusion detection systems: A survey and taxonomy, Tech. Rep. 99-15, Chalmers University (Mar. 2000).

[4] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, K.-Y. Tung, Intrusion detection system: A comprehensive review, Journal of Network and Computer Applications 36 (1) (2013) 16 – 24. doi:http://dx.doi.org/10.1016/j.jnca.2012.09.004.

[5] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, A. D. Keromytis, Casting out demons: Sanitizing training data for anomaly sensors, in: IEEE Symposium on Security and Privacy, 2008. SP 2008., IEEE, 2008, pp. 81–95.

[6] C. Gates, C. Taylor, Challenging the anomaly detection paradigm: a provocative discussion, in: Proceedings of the 2006 workshop on New security paradigms, NSPW '06, ACM, New York, NY, USA, 2006, pp. 21–29.

[7] R. Sommer, V. Paxson, Outside the closed world: On using machine learning for network intrusion detection 0 (2010) 305–316.

[8] S. Forrest, S. A. Hofmeyr, A. Somayaji, T. A. Longstaff, A sense of self for Unix processes, in: Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, 1996, pp. 120–128.

[9] C. Warrender, S. Forrest, B. Pearlmutter, Detecting intrusions using system calls: alternative data models, in: Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, 1999, pp. 133–45. doi:10.1109/SECPRI.1999.766910.

[10] S. Forrest, S. Hofmeyr, A. Somayaji, The evolution of system-call monitoring, in: Computer Security Applications Conference, 2008. ACSAC 2008. Annual, 2008, pp. 418–430.

[11] S. Axelsson, The base-rate fallacy and the difficulty of intrusion detection, ACM Trans. Inf. Syst. Secur. 3 (3) (2000) 186–205. doi:10.1145/357830.357849.

[12] W. Khreich, E. Granger, A. Miri, R. Sabourin, A survey of techniques for incremental learning of HMM parameters, Information Sciences 197 (2012) 105–130.

[13] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM computing surveys (CSUR) 41 (3) (2009) 15.

[14] Y. Liao, V. R. Vemuri, Use of k-nearest neighbor classifier for intrusion detection, Computers & Security 21 (5) (2002) 439–448.

[15] D.-K. Kang, D. Fuller, V. Honavar, Learning classifiers for misuse detection using a bag of system calls representation, Lecture Notes in Computer Science 3495 (2005) 511–516. doi:10.1007/11427995_51.

[16] W.-H. Chen, S.-H. Hsu, H.-P. Shen, Application of SVM and ANN for intrusion detection, Computers & Operations Research 32 (10) (2005) 2617–2634.

[17] S. Rawat, V. Gulati, A. K. Pujari, V. R. Vemuri, Intrusion detection using text processing techniques with a binary-weighted cosine metric, Journal of Information Assurance and Security 1 (1) (2006) 43–50.

[18] A. Sharma, A. K. Pujari, K. K. Paliwal, Intrusion detection using text processing techniques with a kernel based similarity measure, computers & security 26 (7) (2007) 488–495.

[19] G. Salton, Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer, Addison-Wesley, 1989.

[20] G. Creech, J. Hu, Generation of a new ids test dataset: Time to retire the kdd collection, in: Wireless Communications and Networking Conference (WCNC), 2013 IEEE, Shanghai, China, 2013, pp. 4487–4492. doi:10.1109/WCNC.2013.6555301.

[21] W. W. Cohen, Fast effective rule induction, in: A. Prieditis, S. Russell (Eds.), Proc. of the 12$^{\text{th}}$ International Conference on Machine Learning, Morgan Kaufmann, Tahoe City, CA, 1995, pp. 115–123.

[22] W. Lee, D. Xiang, Information-theoretic measures for anomaly detection, in: Proc. of the 2001 IEEE Symposium on Security and Privacy, 2001, pp. 130–143.

[23] W. Fan, M. Miller, S. Stolfo, W. Lee, P. Chan, Using artificial anomalies to detect unknown and known network intrusions, Knowledge and Information Systems 6 (2004) 507–527.

[24] C. C. Michael, A. Ghosh, Simple, state-based approaches to program-based anomaly detection, ACM Trans. Information System Security 5 (2002) 203–237.

[25] R. Sekar, M. Bendre, D. Dhurjati, P. Bollineni, A fast automaton-based method for detecting anomalous program behaviors, in: Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on, 2001, pp. 144–155.

[26] D. Wagner, D. Dean, Intrusion detection via static analysis, in: Proceedings of the 2001 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, 2001.

[27] C. Kruegel, D. Mutz, W. Robertson, F. Valeur, Bayesian event classification for intrusion detection, in: Proceedings of the 19$^{\text{th}}$ Annual Computer Security Applications Conference, ACSAC '03, IEEE Computer Society, Washington, DC, USA, 2003.

[28] S. Jha, K. Tan, R. Maxion, Markov chains, classifiers, and intrusion detection, in: Proceedings of the Computer Security Foundations Workshop, 2001, pp. 206–219.

[29] C. Wressnegger, G. Schwenk, D. Arp, K. Rieck, A close look on n-grams in intrusion detection: anomaly detection vs. classification, in: Proceedings of the 2013 ACM workshop on Artificial intelligence and security, ACM, 2013, pp. 67–76.

[30] C. Marceau, Characterizing the behavior of a program using multiple-length n-grams, in: NSPW '00: Proceedings of the 2000 workshop on New security paradigms, ACM Press, New York, NY, USA, 2000, pp. 101–110. doi:10.1145/366173.366197.

[31] A. Wespi, M. Dacier, H. Debar, Intrusion detection using variable-length audit trail patterns, in: RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, Springer-Verlag, London, UK, 2000, pp. 110–129.

[32] W. Khreich, E. Granger, R. Sabourin, A. Miri, Combining Hidden Markov Models for anomaly detection, in: International Conference on Communications (ICC), Dresden, Germany, 2009, pp. 1–6.

[33] J. Hu, Host-based anomaly intrusion detection, in: P. Stavroulakis, M. Stamp (Eds.), Handbook of Information and Communication Security, Springer Berlin Heidelberg, 2010, pp. 235–255. doi:10.1007/978-3-642-04117-4_13.

[34] L. E. Baum, G. S. Petrie, N. Weiss, A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains, The Annals of Mathematical Statistics 41 (1) (1970) 164–171.

[35] C. D. Manning, P. Raghavan, H. Schütze, Introduction to information retrieval, Vol. 1, Cambridge university press Cambridge, 2008.

[36] K. Rieck, T. Holz, C. Willems, P. Düssel, P. Laskov, Learning and classification of malware behavior, in: Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2008, pp. 108–125.

[37] Y. Liao, V. R. Vemuri, Using text categorization techniques for intrusion detection., in: USENIX Security Symposium, Vol. 12, 2002.

[38] D.-K. Kang, D. Fuller, V. Honavar, Learning classifiers for misuse and anomaly detection using a bag of system calls representation, in: Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE, 2005, pp. 118–125. doi:10.1109/IAW.2005.1495942.

[39] G. Creech, J. Hu, A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns, IEEE Transactions on Computers 99 (2013) –. doi:10.1109/TC.2013.13.

[40] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: Theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.

[41] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, Optimization of sparse matrix–vector multiplication on emerging multicore platforms, Parallel Computing 35 (3) (2009) 178–194.

[42] K. M. C. Tan, K. S. Killourhy, R. A. Maxion, Undermining an anomaly-based intrusion detection system using common exploits, In Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002). Lecture Notes in Computer Science, Springer-Verlag, Berlin 2516 (2002) 54–73.

[43] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt, Support vector method for novelty detection., in: NIPS, Vol. 12, 1999, pp. 582–588.

[44] D. M. Tax, R. P. Duin, Support vector data description, Machine learning 54 (1) (2004) 45–66.

[45] T. Fawcett, An introduction to ROC analysis, Pattern Recogn. Lett. 27 (8) (2006) 861–874. doi:10.1016/j.patrec.2005.10.010.

[46] J. Huang, C. Ling, Using auc and accuracy in evaluating learning algorithms, IEEE Transactions on Knowledge and Data Engineering 17 (3) (2005) 299–310. doi:10.1109/TKDE.2005.50.

[47] J. Hanley, B. McNeil, The meaning and use of the area under a receiver operating characteristic (roc) curve, Radiology 143 (1) (1982) 29–36.

[48] S. D. Walter, The partial area under the summary roc curve, Statistics in Medicine 24 (13) (2005) 2025–2040. doi:10.1002/sim.2103.

[49] I. Corona, G. Giacinto, F. Roli, Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues, Inf. Sci. 239 (2013) 201–225. doi:10.1016/j.ins.2013.03.022.

[50] D. Wagner, P. Soto, Mimicry attacks on host-based intrusion detection systems, in: CCS '02: Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, United States, 2002, pp. 255–264. doi:10.1145/586110.586145.

[51] C. Kruegel, D. Mutz, F. Valeur, G. Vigna, On the detection of anomalous system call arguments, in: E. Snekkenes, D. Gollmann (Eds.), Computer Security – ESORICS 2003, Vol. 2808 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003, pp. 326–343. doi:10.1007/978-3-540-39650-5_19.

[52] F. Maggi, M. Matteucci, S. Zanero, Detecting intrusions through system call sequence and argument analysis, IEEE Transactions on Dependable and Secure Computing, 7 (4) (2010) 381–395. doi:10.1109/TDSC.2008.69.

[53] N. Provos, Improving host security with system call policies., in: USENIX Security, Vol. 3, 2003.

[54] G. Tandon, P. K. Chan, Learning rules from system call arguments and sequences for anomaly detection, in: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM) Workshop on Data Mining for Computer Security (DMSEC), Melbourne, Florida, USA, 2003.

[55] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, W. Gong, Anomaly detection using call stack information, in: Security and Privacy, 2003. Proceedings. 2003 Symposium on, 2003, pp. 62–75.

[56] D. Gao, M. K. Reiter, D. Song, On gray-box program tracking for anomaly detection, in: Proceedings of the 13th conference on USENIX Security Symposium - Volume 13, SSYM'04, USENIX Association, Berkeley, CA, USA, 2004, pp. 1–8.

[57] D. Gao, M. K. Reiter, D. Song, Gray-box extraction of execution graphs for anomaly detection, in: Proceedings of the 11th ACM conference on Computer and communications security, ACM, 2004, pp. 318–329.

[58] D. Mutz, W. Robertson, G. Vigna, R. Kemmerer, Exploiting execution context for the detection of anomalous system calls, in: Recent Advances in Intrusion Detection, Springer, 2007, pp. 1–20.

[59] K. Tan, R. Maxion, "Why 6?" Defining the operational limits of stide, an anomaly-based intrusion detector, in: IEEE Symposium on Security and Privacy, 2002, pp. 188–201. doi:10.1109/SECPRI.2002.1004371.

[60] J. Han, Q. Yan, R. Deng, D. Gao, On detection of erratic arguments, in: M. Rajarajan, F. Piper, H. Wang, G. Kesidis (Eds.), Security and Privacy in Communication Networks, Vol. 96 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, 2012, pp. 172–189. doi:10.1007/978-3-642-31909-9_10.

[61] H. H. Feng, J. T. Giffin, Y. Huang, S. Jha, W. Lee, B. P. Miller, Formalizing sensitivity in static analysis for intrusion detection, in: Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on, IEEE, 2004, pp. 194–208.

[62] J. T. Giffin, S. Jha, B. P. Miller, Detecting manipulated remote call streams., in: USENIX Security Symposium, 2002, pp. 61–79.

[63] J. T. Giffin, S. Jha, B. P. Miller, Efficient context-sensitive intrusion detection, in: Proceedings of the Network and Distributed System Security Symposium, 2004.

[64] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, G. Vigna, Automating mimicry attacks using static binary analysis, in: Proceedings of Security '05, the 14$^{\text{th}}$ USENIX Security Symposium, Baltimore, MD, USA, 2005, pp. 161–176.

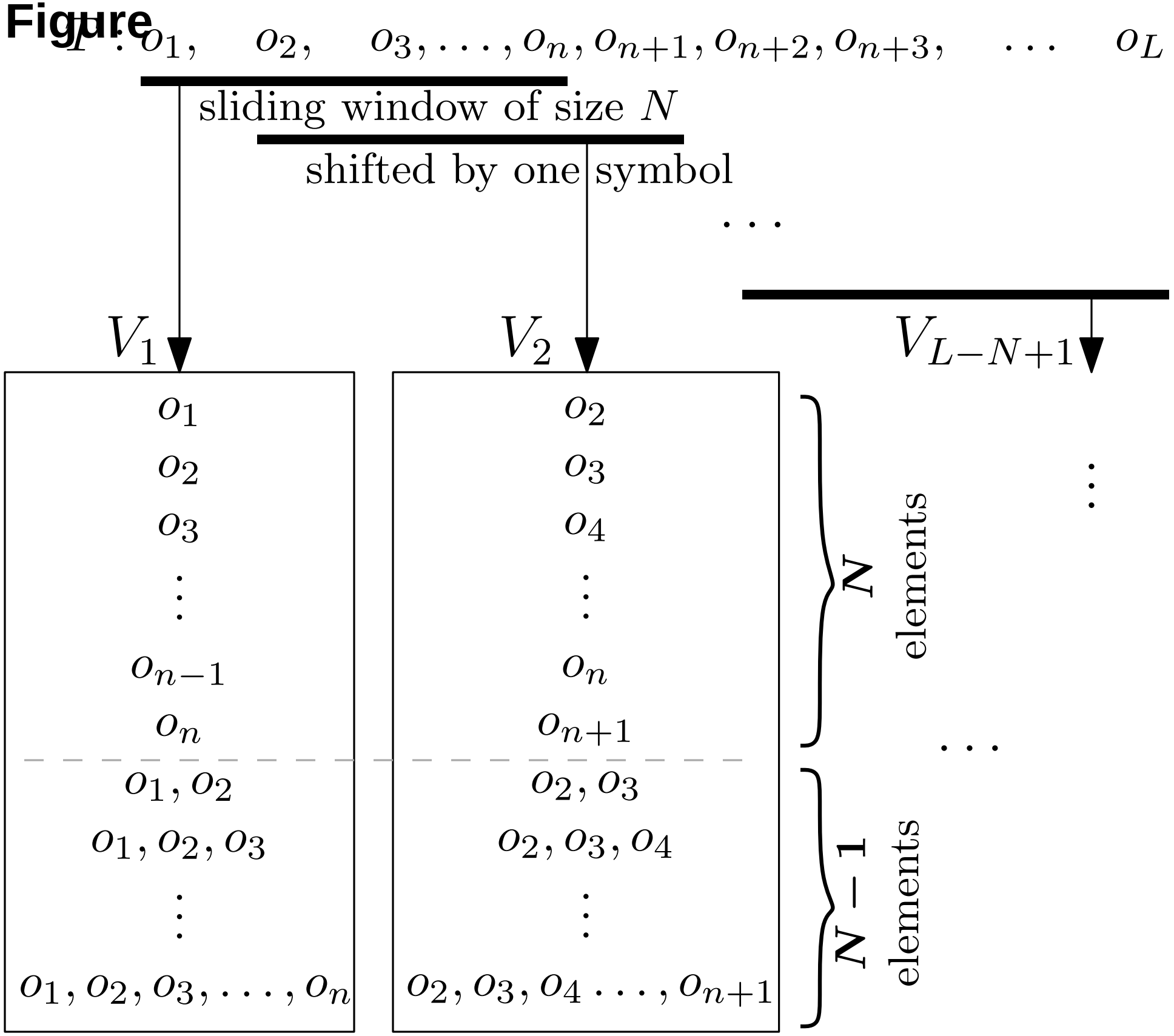
[65] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, R. K. Iyer, Non-control-data attacks are realistic threats, in: Proceedings of the 14th conference on USENIX Security Symposium, Vol. 14, 2005, pp. 12–12.

[66] S. Bhatkar, A. Chaturvedi, R. Sekar, Dataflow anomaly detection, in: IEEE Symposium on Security and Privacy, 2006. doi:10.1109/SP.2006.12.

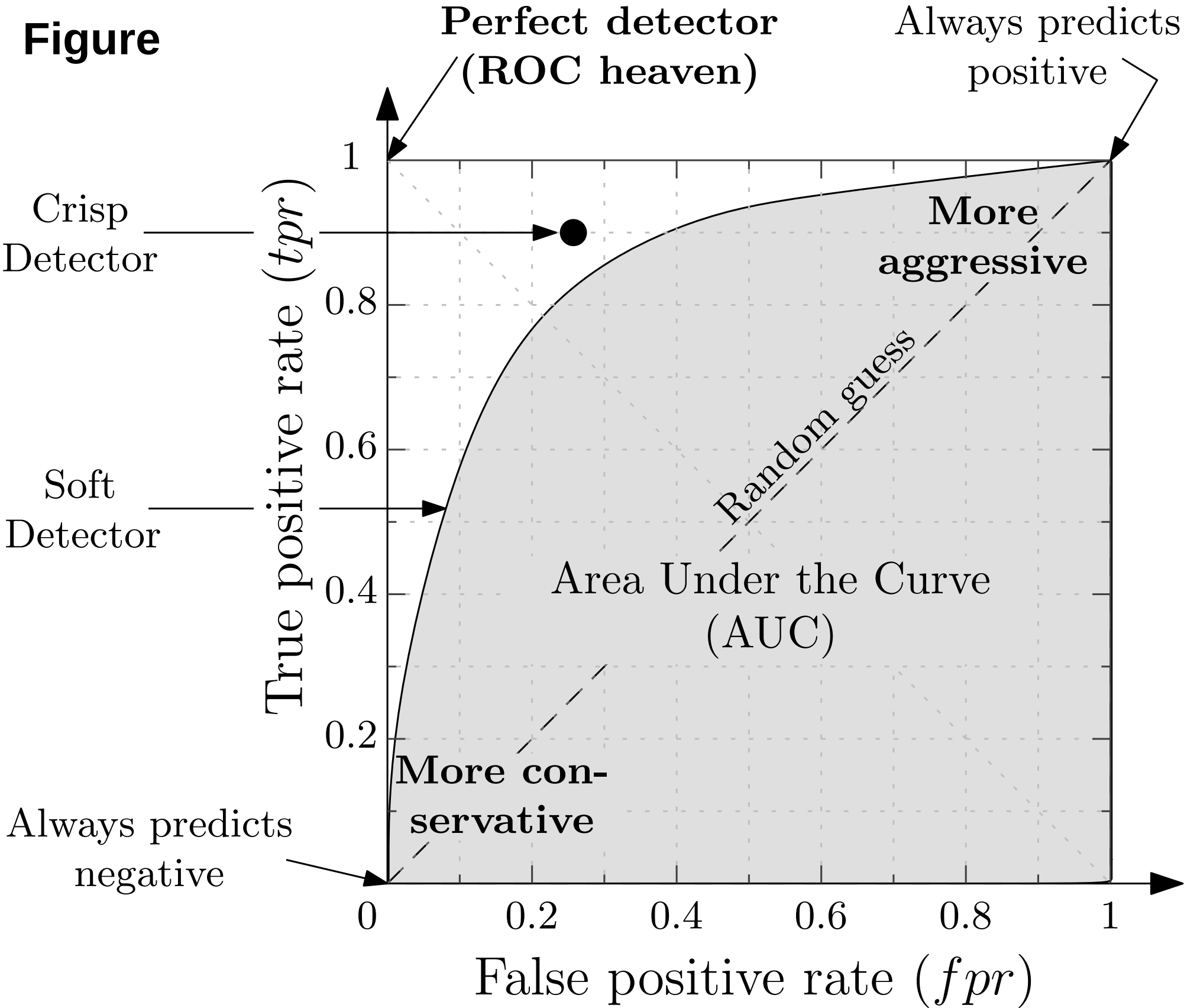
[67] C. Parampalli, R. Sekar, R. Johnson, A practical mimicry attack against powerful system-call monitors, in: Proceedings of the 2008 ACM symposium on Information, computer and communications security, ASIACCS '08, ACM, New York, NY, USA, 2008, pp. 156–167.

[68] H. Kayacik, A. Zincir-Heywood, Mimicry attacks demystified: What can attackers do to evade detection?, in: Privacy, Security and Trust, 2008. PST '08. Sixth Annual Conference on, 2008, pp. 213–223. doi:10.1109/PST.2008.25.

Figure


$L: o_1, \quad o_2, \quad o_3, \ldots, o_n, o_{n+1}, o_{n+2}, o_{n+3}, \quad \ldots \quad o_L$
sliding window of size N
shifted by one symbol
$\ldots$
V_1
V_2
V_{L-N+1}
o_1 o_2 o_3 $\vdots$ o_{n-1} o_n
o_2 o_3 o_4 $\vdots$ o_n o_{n+1}
$\vdots$
N elements
$\ldots$
o_1, o_2 o_1, o_2, o_3 $\vdots$ $o_1, o_2, o_3, \ldots, o_n$
o_2, o_3 o_2, o_3, o_4 $\vdots$ $o_2, o_3, o_4 \ldots, o_{n+1}$
$N-1$ elements

Figure


Perfect detector
(ROC heaven)
Always predicts
positive
Crisp
Detector
Soft
Detector
More
aggressive
Random guess
Area Under the Curve
(AUC)
More con-
servative
Always predicts
negative
True positive rate (tpr)
False positive rate (fpr)
0
0.2
0.4
0.6
0.8
1

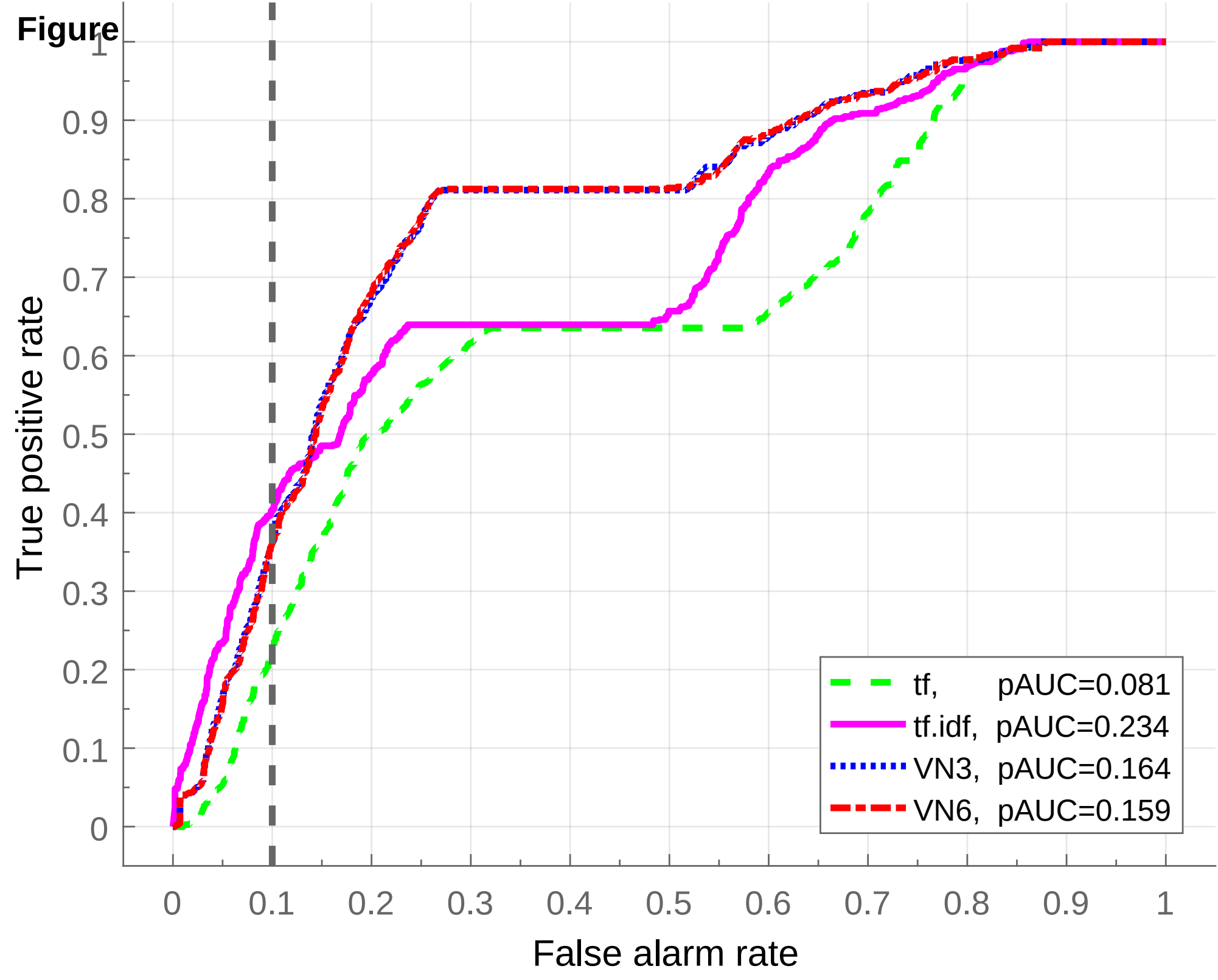


Figure
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
tf, pAUC=0.081
tf.idf, pAUC=0.234
VN3, pAUC=0.164
VN6, pAUC=0.159

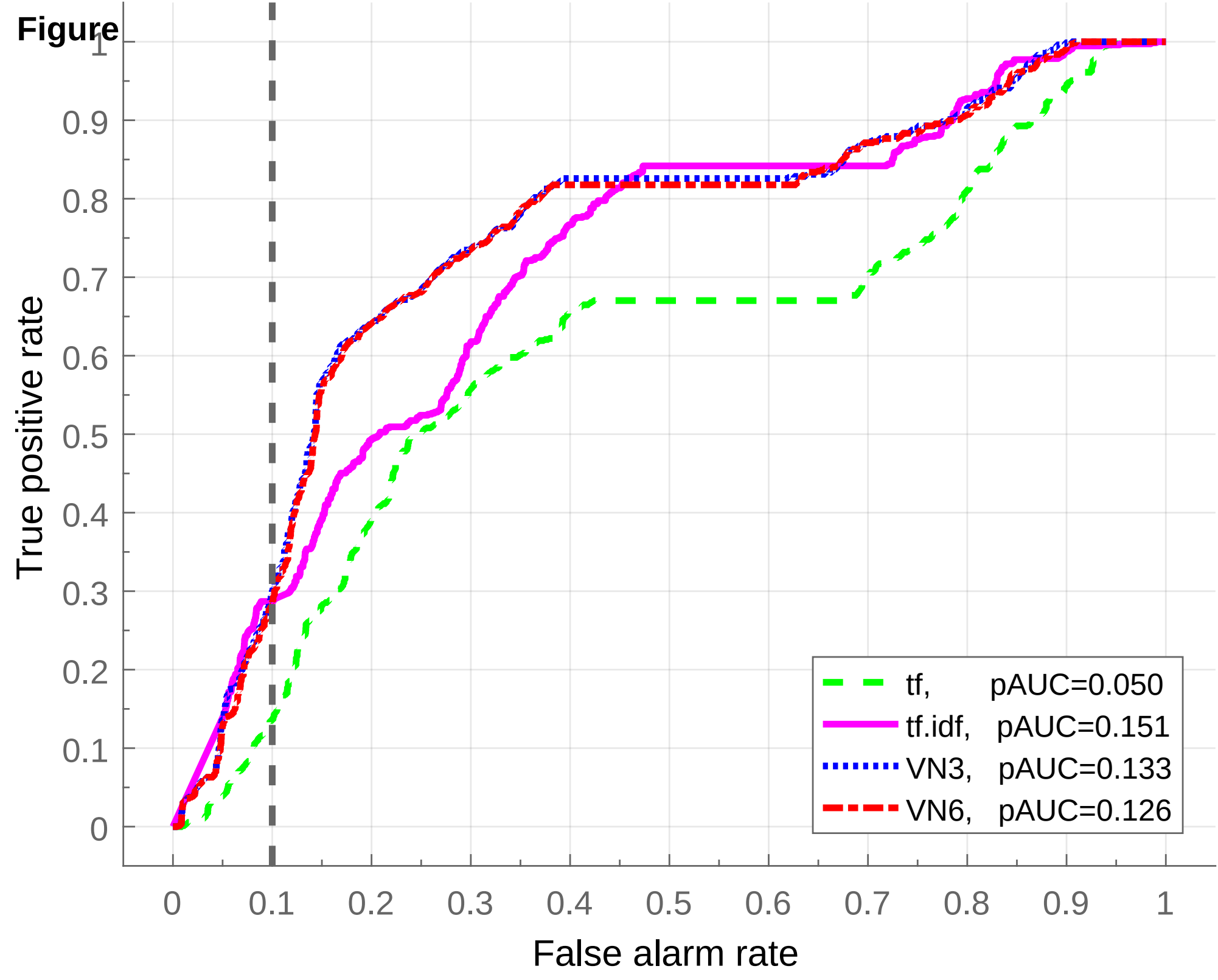


Figure
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
tf, pAUC=0.050
tf.idf, pAUC=0.151
VN3, pAUC=0.133
VN6, pAUC=0.126

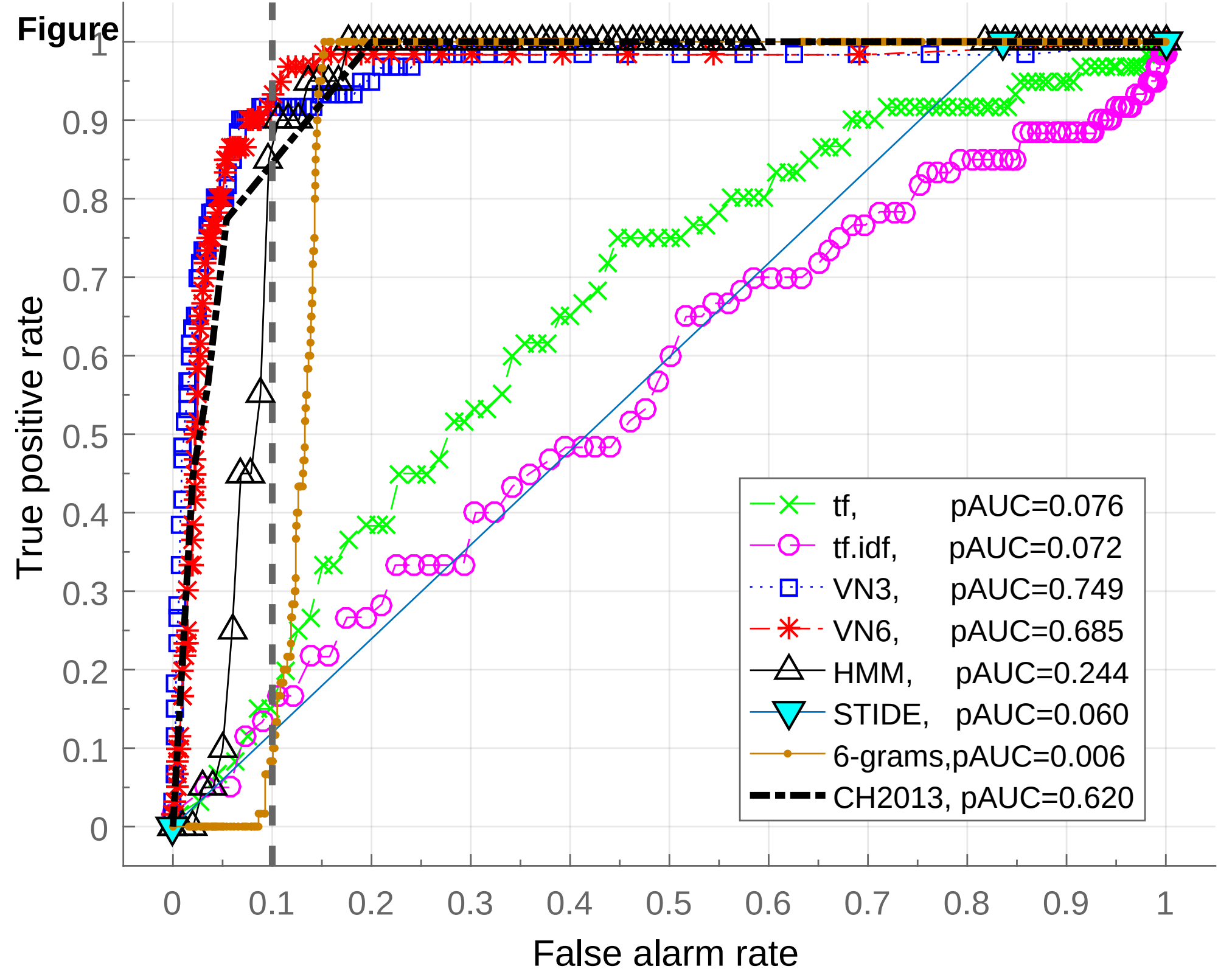


Figure
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
tf, pAUC=0.076
tf.idf, pAUC=0.072
VN3, pAUC=0.749
VN6, pAUC=0.685
HMM, pAUC=0.244
STIDE, pAUC=0.060
6-grams,pAUC=0.006
CH2013, pAUC=0.620

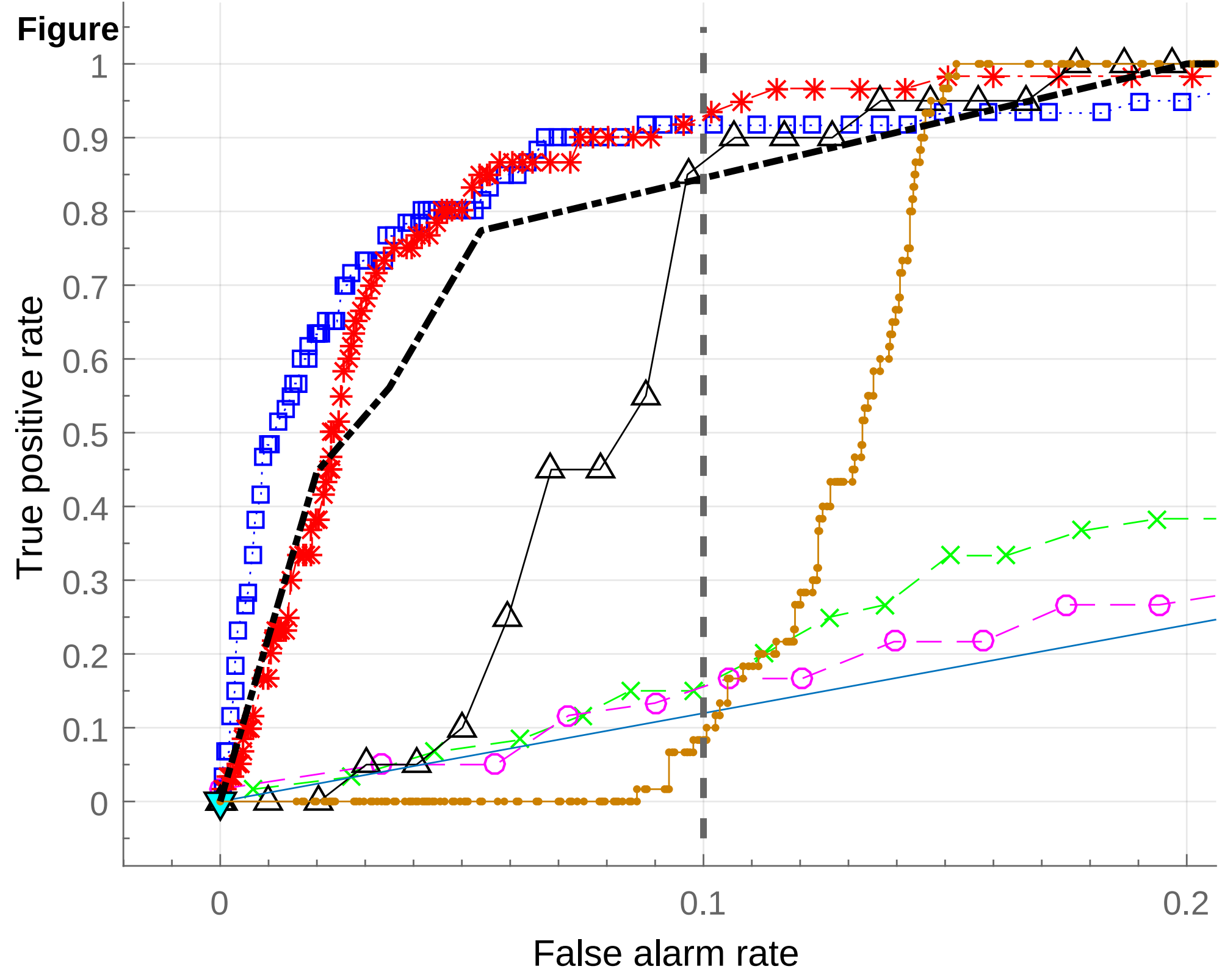


Figure
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.1
0.2

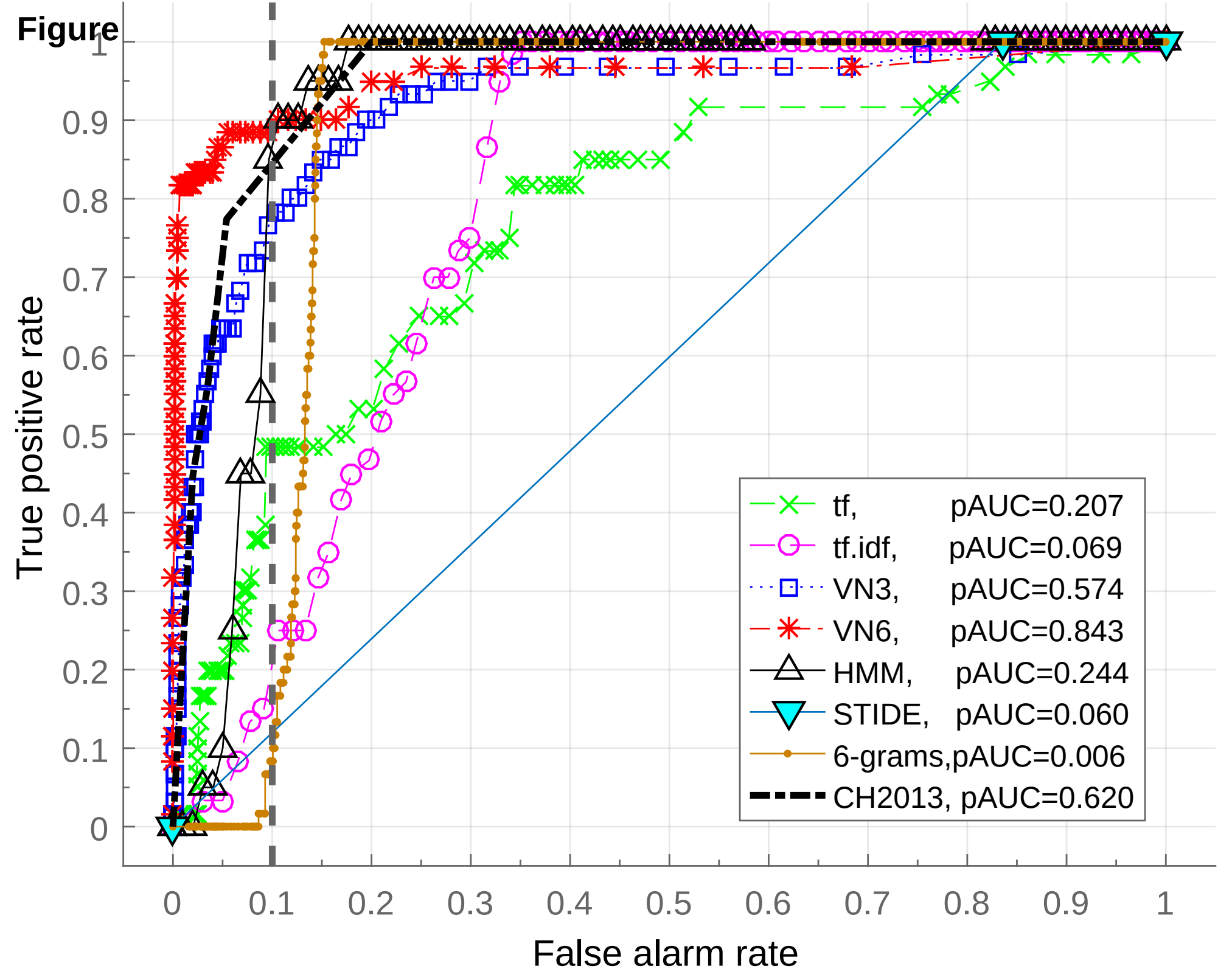


Figure
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
tf, pAUC=0.207
tf.idf, pAUC=0.069
VN3, pAUC=0.574
VN6, pAUC=0.843
HMM, pAUC=0.244
STIDE, pAUC=0.060
6-grams,pAUC=0.006
CH2013, pAUC=0.620

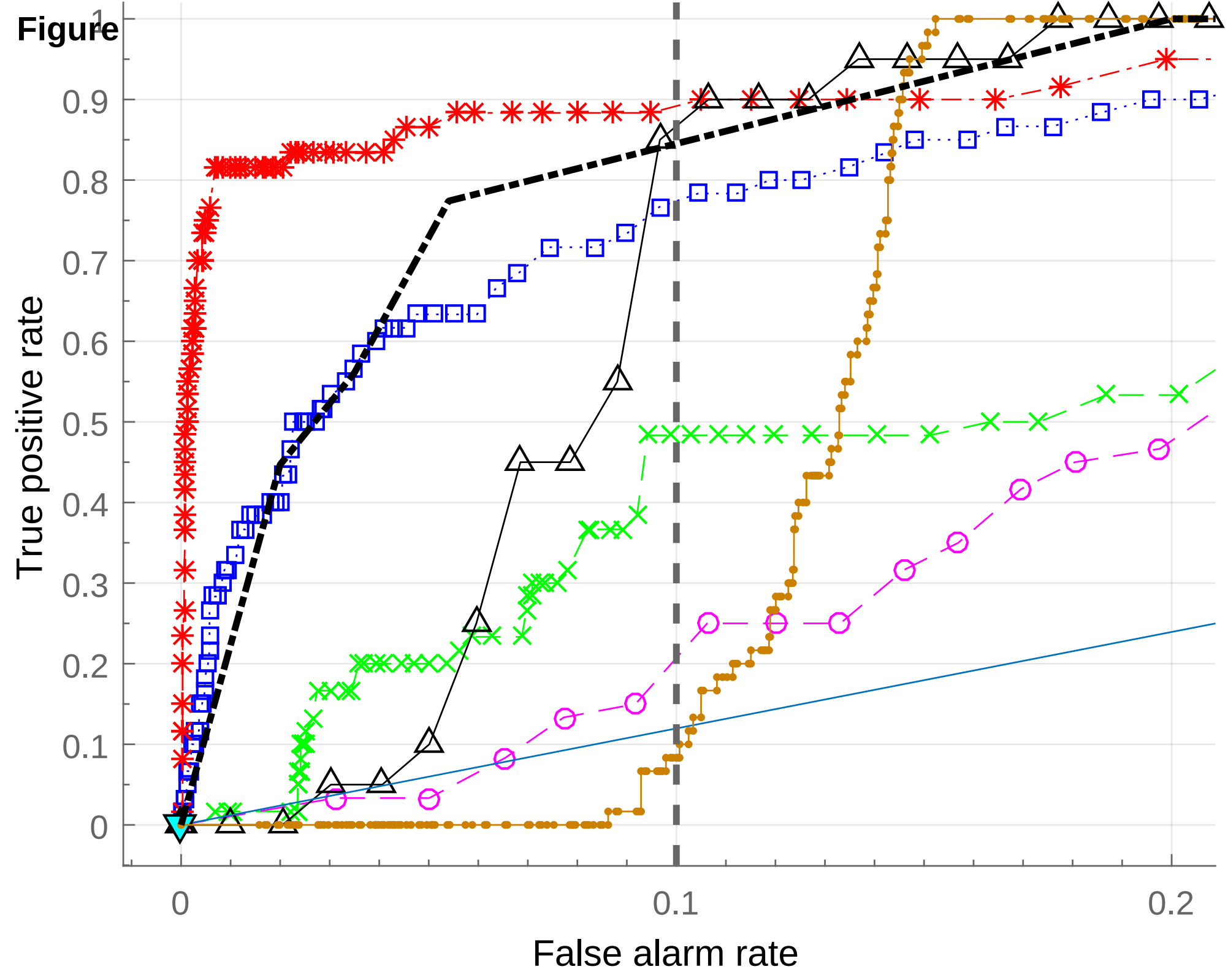


Figure
True positive rate
False alarm rate
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
0.1
0.2

LaTeX Source Files

Click here to download LaTeX Source Files: ist17.tex