Intrusion Detection in Unstructured Contexts Using On-line Clustering and Novelty Detection

Eduardo Alves Ferreira [1]
Rodrigo Fernandes de Mello [1]

Abstract: The characterization of processes behavior is usually considered when performing intrusion detection. Several works characterize specific aspects of systems and attempt to detect novelties in that context, associating observed anomalies to attack events. Such approach is limited or even useless when the observed context is unstructured, i.e. when the monitor generates text-based log files or a variable number of application attributes. In order to overcome such drawback, this paper considers the use of single-pass clustering techniques to apply a quantization operation to unstructured data and generate time series, using algorithms with low computational complexity, applicable in a real-world scenario. Afterward, novelty detection techniques are employed on such series to distinguish behavior anomalies, which are associated with intrusions. We evaluated the approach using a system characterization dataset and confirmed that it aggregates context information to represent the behavior of applications as time series, where novelty detection can be successfully performed.

1 Introduction

The characterization of process behavior is commonly considered when performing autonomic intrusion detection. Using this approach, one first extracts and characterizes the normal behavior of a system (e.g. when an application is working in an isolated network, in a simulated environment or during acceptance or stress testing sessions). Later on, the system is monitored in production environment, considering observed novelties as possible attack incidents.

To characterize the behavior of an application, three approaches are described in related works: The first one uses model-based techniques to evaluate application outputs, ignoring any temporal dependency among events. The second approach considers only the temporal dependency, among a limited number of observations. The major limitations of those approaches are the fact that they ignore one aspect of the application, and also the fact that proposed solutions are closely tied to a specific application domain, hindering the proposal of a fully autonomous solution. A third approach, based on clustering and time series analysis, is not only capable of addressing both aspects of an application (i.e. the multi-dimensionality of system observations and the temporal dependency among objects), but can

[1] Institute of Mathematical and Computer Sciences, USP
{eaf,mello@icmc.usp.br}

also be independent of the monitored data model. Most works [1, 2] that follow this third approach present a major shortcoming, though: they are based in batch approaches, using algorithms with high computational complexity that must sweep a dataset several times, which is not acceptable in a real-world scenario.

In this paper, we claim that the use of clustering in conjunction with novelty detection in order to find intrusion events is applicable to production systems, in which no secondary storage is available to maintain all monitored data, forcing the use of single-sweep approaches. We evaluate the use of single-pass clustering techniques to apply a quantization operation to unstructured data, allowing the characterization of complex application behavior with little knowledge about its structure. We also evaluate the consistency of this approach analyzing the correlation between the indexes of clustering quality and the capacity of differentiation between normal (i.e. regular usage) and attack situations.

We validate this approach by applying several clustering and novelty detection algorithms on a dataset of system behavior characterization [3]. We confirmed that the precision of single-pass clustering and novelty detection algorithms is comparable to the one obtained with batch algorithms. We then check the correlation in between the clustering validation indexes (quadratic error and silhouette) and the detection results, finding that the greater differentiation in between normal and attack situations occurs when we have better clustering indexes. This confirms the hypothesis that, having better clustering, the generated time series contains more information, what tends to provide a better characterization of systems behavior. Also, since the clustering indexes indicate the quality of the final solution, they can be used to tailor the definition of our approach parameters on a real scenario. Finally, we present Receiver Operating Characteristic (ROC) curves obtained after testing our attack detection approach. We also analyze the area under the ROC curves with regard to the obtained clustering indexes, also observing that better clustering indexes yield in greater characterization capacity.

This paper is organized as follows: Section 2 presents related work; Section 3 describes the proposed approach and how algorithms were applied on the system characterization dataset; Experimental results are presented in Section 4; Section 5 presents concluding remarks and, finally, references.

2 Related Work

Two areas of Computer Science comprise the fundamental basis for this work: clustering [4] and novelty detection [5]. This work has also a strong data stream bias [6], since we are interested in characterizing the behavior of systems under the production scenario, where system behavior data is continuously produced over time, and no secondary storage is available to maintain all data at once. In this context, we need algorithms that not only have

near-linear time complexity, but also operate on the dataset in a single "sweep", i.e. perform on every observation at the time they are produced (or very closely to that), keeping only a small amount of data into memory. Given such assumptions, we firstly present some clustering algorithms that satisfy such needs, to later present some novelty detection approaches that can be applied to this problem.

2.1 Single-sweep Clustering

Clustering algorithms are categorized as either hierarchical or partitional [4]. Most hierarchical algorithms present high time complexity, therefore they are not applicable in the context of this work. Several partitional algorithms present near-linear complexity, although not all of them are capable of working in a single-sweep fashion. Some single-sweep incremental algorithms are then evaluated in this work, and the results are compared to the results of K-Means, a well-known partitional algorithm.

One of those on-line algorithms is the Leader-Follower [7], which maintains a set of centroids in memory, to represent the most usual behavior of dataset observations. A parameter θ is assigned to every centroid, to represent its coverage area. The algorithm sweeps the dataset and, for every object, computes the distance from the object to every centroid. The centroid with the smallest distance to the observed object is considered the winner. If the distance in between the object and the winning centroid is smaller than θ, then the object is associated with that centroid, otherwise a new centroid is created at the position of the observed object. Using a simplistic approach, the value of θ is the same for all centroids and it does not change as objects are observed. Also, after a centroid is created, its position is never moved. A more sophisticated approach moves the centroid as objects are added, using an average of them [7]. In this paper, in order to distinguish both approaches, we name this latter as "On-line K-Means Leader-Follower".

Some of the limitations of the Leader-Follower approach are the need for defining the parameter θ, and the fast growth of the number of centroids when θ is underestimated. In order to deal with these limitations, the adaptive Leader-Follower algorithm was proposed, known as Greedy Leader-Follower Adaptive algorithm [8]. This algorithm limits the maximum number of available centroids: if the number of centroids exceeds a parameter k, an operation is executed to reduce such number, merging the two closest centroids into a single one.

Another adaptive clustering algorithm evaluated in this work is the "Grow When Required" (GWR) [9] artificial neural network. GWR adds new centroids when none of the existing ones is capable of representing an observation with enough precision. It is also capable of removing centroids when their activity is too low (i.e. when they are no longer required). GWR is not an on-line algorithm, since it sweeps a dataset through t iterations.

Another limitation is that it depends on a centroid adaptation function based on the weighted mean among attributes. An adaptation of this algorithm is proposed in this work not only to apply it in an on-line fashion, but also to allow the use of datasets with non-numerical attributes.

In order to evaluate the precision of these on-line algorithms, a comparison of their results against the K-Means [10] was conducted. This comparison was conducted using the mean quadratic error (Q.E.) [4] and the silhouette [11]. Since the traditional silhouette cannot be used in an on-line fashion, we use the simplified silhouette [12] as an approximation for this value.

2.2 Novelty Detection

Several techniques have been used to characterize the behavior of UNIX processes as time series. They usually extract a unidimensional aspect of the application, composing time series that represent the behavior when performing operational tasks. Then, a training stage is required to characterize the normal application behavior, i.e., the behavior on an attack-free environment. Later on, the system is installed in the production environment and novelty detection techniques are applied to detect any attack traces.

(author?) [1] present a sliding window approach to detect novelties in sequences of system calls. This technique is based on supervised learning, with background on computational immunology. At the training stage, time series are evaluated within a sliding window, and all observed sequences are stored in memory. A prefix tree is applied to increase the search efficiency and reduce storage requirements. When the system is then executed in a production environment, the observed windows are compared to the sequences stored in memory. The amount of novelty at that point is defined as the smallest Hamming distance [13] between the observed window and every window in the tree. The division of this Hamming distance by the window length results in a normalized amount of novelty per object.

Another approach is considered by **(author?)** [14], in which Shannon's Entropy [15] of series is used as novelty index. Each observation in the sequence is represented as a state on a Markov Chain, and the probability of transition in this chain is estimated from the sequences in the time series. The amount of novelty in the series at a specific time is given by the derivative of the Entropy with respect to time. We also expect to find similar Entropy values when we compare similar series, so, in this paper, the Entropy of two series is also used to differentiate sequences.

Finally, we consider the Dynamic Time Warping (DTW) distance to compare series. DTW is a distance function that is not sensitive to out-of-phase signals. While presenting good results for the characterization of process behavior [14], this technique is not useful for the proposed problem due to its quadratic time complexity regarding the series length.

However, it is useful as baseline for comparison purposes.

2.3 System Characterization Dataset

Several system characterization datasets are presented or described in the literature. By far, the most widely used is the "1998 and 1999 DARPA intrusion detection evaluation", which contains network traffic and Solaris BSM output captured during several weeks on a simulated environment. However, this dataset is not useful in the context of this paper mainly due to two reasons: first, several flaws [16] are known to be present on this dataset, and second, most of the information present on this dataset comes from network sessions: the only data regarding process characterization comes from BSM logs, and such records do not contain every system call executed by processes under attack, what is assumed to be essential in our work.

(author?) [17] and **(author?)** [18] present and describe valid system characterization datasets, with hundreds of thousands of system calls extracted from production systems. However, either the system call parameters are not present or the datasets are not published because they contain sensitive information, which is a common issue faced when data is extracted from production systems.

On the other hand, **(author?)** [3] provide a dataset of system characterization which contains system calls and their parameters for `WU-FTPD`, an FTP server. To build this dataset, a public domain dataset of FTP protocol sessions was used to simulate 55 sessions of normal usage, while attack sessions were manually executed. Although this dataset is shorter than others (containing around $85,000$ system calls, when they usually have around a million objects) it is more appropriate for this work since it is public and contains the value of system call parameters. Four successful attack sessions are presented in this dataset. The first one represents an intrusion that obtains the root shell, but that does not execute any command after the access is granted. The second and third attack sessions simulate actions that would usually be executed during an attack after the administrative access was obtained. The fourth session obtains root access and executes an FTP bounce attack against another host, which is expected to generate a lot of anomalous behavior. These three classes of attack are named, in this paper, as A, B and C, respectively.

3 Proposed Approach

In this paper, we propose the use of single-sweep clustering and novelty detection algorithms to represent the application behavior and support intrusion detection in unstructured contexts. The quantization stage generates a sequence of identifiers from unstructured objects, which represents the temporal behavior of an application. This sequence is then

submitted to novelty detection algorithms to distinguish intrusion events.

In order to meet this objective, we employ the architecture presented in Figure 1. Such architecture is based on three stages: the first one (a) extracts the objects or events from a system under the production environment (e.g. system call data, CPU and memory usage, log or network messages), providing the data or domain model; such data model is input to the next stage (b), which considers a distance function to characterize the similarity among objects (also named observations or events), an optional merge function (which is responsible for representing several objects as a single one), and also a clustering algorithm that receives objects as input and finally produces time series representing the system behavior; such time series is input to the last stage (c), that applies a novelty detection algorithm on it in order to distinguish attack events (novelties) from the normal behavior.

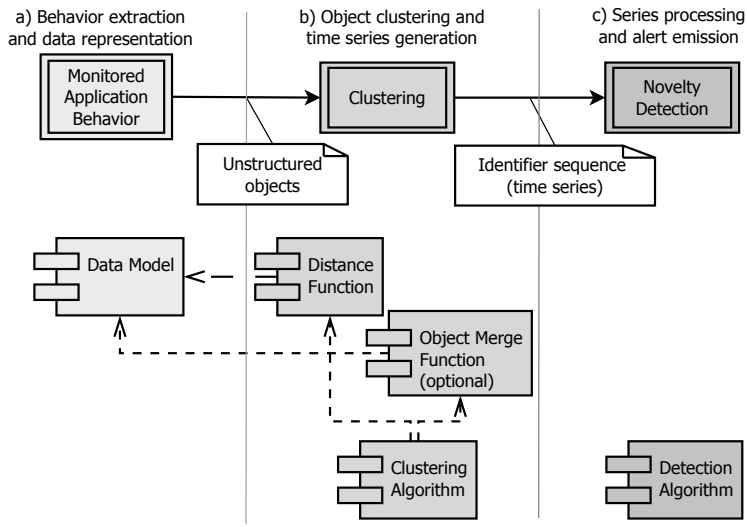


Figure 1. Clustering and Novelty Detection Architecture.

To validate this approach, we apply several clustering and novelty detection algorithms on the intrusion dataset provided by **(author?)** [3]. In the first step, the approach reads text log files, parsing system call instances into objects. Each object contains the system call type, a list with all parameter values and, when available, the return value. All those attributes characterize an object and consecutive objects are used as input to clustering algorithms. Those algorithms require a manner of comparing the relative similarities of objects, what is done by using a distance function. In that sense, we defined a hierarchical-based distance function. The first step of this function is to group system calls under the same type. In this way, we guarantee that when we have the worst clustering performance (i.e. when all objects

are clustered in a single group) we produce series that represent at least the system call type. Under a better clustering performance, we add more information to the generated time series, this is, series observations represent not only the system call type but also parameters and return values.

After defining the distance function, we selected a set of clustering algorithms to be evaluated for the proposed scenario. The selected algorithms are K-Means [10], Simple Leader-Follower [7], On-line-K-Means Leader-Follower [7], Adaptive Leader-Follower [8] and the GWR artificial neural network [9]. All of them are on-line algorithms or were applied in an on-line fashion, except K-Means. K-Means was chosen as a baseline of comparison because it is proven to converge to local optimum solutions after few iterations. In order to increase the chances of finding a global maximum, we executed 16 random initializations for K-Means and iterations were run until the value of the quadratic error stabilized in a local minimum. Moreover, to increase the precision of the clustering indexes, we performed two sweeps on the database following the order in which the data was generated. The first sweep is conducted as training stage, where the centroids are created and adapted according to dataset observations. The second sweep is a validation stage, in which we look for the closest centroids to every observation and estimate the clustering indexes from these distances.

The Leader-Follower algorithm was tested in three variations: Simple, On-line-K-Means and Adaptive. The simple approach uses a constant parameter θ, and centroids do not move/adapt according to new objects. The On-line-K-Means approach is similar to the previous approach, having no parameter adaptation, but it uses the average of all objects clustered in a group as centroid. The adaptive approach is based on the Greedy algorithm [8], which is initialized with a low value for parameter θ but also with a fixed value for parameter k that limits the maximum number of allowed centroids. When the number of centroids exceeds k, a reduction operation takes place to decrease the total number of centroids. This reduction operation merges the two closest centroids as a single one. In our implementation, the new centroid is not an average of the other two, but it will just keep the position of the centroid with greater diameter. GWR artificial neural network [9] was extended to support on-line learning and also non-numerical attributes.

The clustering validation indexes, quadratic error and silhouette, are also estimated in an on-line fashion, during a second sweep of the dataset. For each point we estimate two values: the distance from the point to centroid where it is placed, the distance between this point an the second closest centroid. The quadratic error index is given by the arithmetic mean of the first distance, for all of the points on the dataset. As an internal clustering validation index, the value of the quadratic error always decreases when the number of groups is increased, since the intra-group dispersion is minimized when more points are used to represent the dataset. The silhouette value, on the other hand, considers not only the intra-group

dispersion, but also the separation among groups. The simplified silhouette value is given by an average that considers both distances. This value represents, through a normalized index, how well a given point is placed in its cluster, considering the distance from every point to the second-closest centroid and the distance to the centroid of the group where it was placed. The average value ranges from -1 to 1, where greater values indicate better clustering.

Three novelty detection approaches were considered in our study: Sliding Window [1], Markov Chain Entropy [14] and Dynamic Time Warping [2]. They take one time series as input in a training stage, and then evaluate the amount of novelty found on new observations. Normal behavior series are used at the characterization phase (training), while testing is performed with both: normal usage and attack series. It is expected that techniques find a small amount of novelty when exposed to the normal behavior, and high amount when assessing attack sessions.

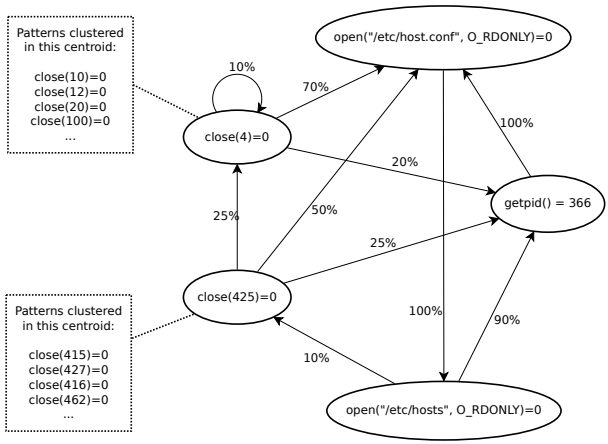


Figure 2. Markov chain with outputs from clustering stage as states.

For the Sliding Window technique, the amount of novelty found in a sequence is then used as index to differentiate two series. To estimate the Shannon's Entropy of a sequence, we define a Markov Chain in which the states represent the centroids obtained during the clustering stage (i.e. the observations from the time series) and the transition probabilities are estimated based on the sequence of the observations on the dataset, as shown in Figure 2. We also considered the Dynamic Time Warping (DTW) to quantify the amount of novelty in

a series, a batch technique that does not meet the constraints of our problem, but is employed only for comparison purposes. DTW uses a dynamic programming algorithm similar to the Levenshtein distance algorithm [**?**], taking two series as input to compute the distance in between them.

4 Experiments

In this section, we present the results of experiments where the proposed approach was applied to the system call dataset proposed by **(author?)** [3]. We first describe the clustering results and then we discuss the results of the novelty detection stage. We also look for the correlation in between clustering and novelty detection results, considering the hypothesis that series with better clustering performance contain greater amount of information and, therefore, better characterize the system behavior. Finally, we present and analyze ROC curves estimated from the observed novelty levels.

4.1 Clustering Results

In order to evaluate clustering results, we considered two indexes: the mean quadratic error and the silhouette (as explained in Section 3). Figure 3 presents the clustering results. We can notice that K-Means is more efficient than the other algorithms (i.e. presents lower values of quadratic error and higher values of silhouette) only when the number of clusters is low. The on-line algorithms present a similar clustering performance to the K-Means algorithm, specially for greater number of centroids. This happens because K-Means is a more sophisticated algorithm, splitting the dataset in groups that are more significant than the other algorithms, a feature that is important when the number of groups is small. On the other hand, when the number of centroids is large, all groups have a reduced size, and this feature becomes less significant, at least for the proposed dataset. Analyzing the silhouette we may notice that Greedy Leader-Follower and GWR present similar performance to K-Means, even for a reduced number centroids. This happens due to the fact that these algorithms are capable of re-evaluating the centroids, while the others are punished when the initially selected centroid points are not good representants of the dataset.

Considering the clustering performance indexes and the ease of use of the algorithms, we observe that Adaptive Leader-Follower shows good performance indexes, having parameters that are much easier to estimate than parameter θ for the simple Leader-Follower and On-line-K-Means Leader-Follower. In order to use this algorithm, we only need to define an underestimated initial θ and an overestimated k. Moreover, by taking the Adaptive Leader-Follower into account, we also avoid scalability issues observed in all other on-line algorithms, where the incorrect estimation of parameters would result in the consumption of huge

amounts of memory and CPU cycles. GWR, on the other hand, demands the definition of the "insertion threshold" parameter, which is very similar to the θ of the Leader-Follower (i.e. a centroid coverage region), and, as such, it is considerably hard to estimate without a prior exploratory study of the dataset. Another important aspect is that the Adaptive Leader-Follower algorithm does not demand the definition of a point-merging function, what might not be available in some domains. That is why this latter algorithm is more appropriate for building autonomous solutions.

4.2 Novelty Detection Results

In this section we discuss the outputs of the novelty detection techniques and how this information is used to distinguish attacks from normal sessions. All novelty detection techniques take as input series that represent the behavior of the system in an attack-free environment, and later evaluate the novelty found on other sequences. The amount of novelty observed in a sequence is then used as an indication of attack probability. For instance, Tables 1 and 2 are samples of the output of the detection stage, presenting the results of the application of DTW to series generated by the clustering algorithm K-Means, with $k = 1$ (i.e. when all the system calls of the same type are clustered in a single group, regardless of their parameter values) and $k = 16$ (i.e. aggregating a higher amount of information). In all these tables we notice that the average novelty level in attack situations is higher than in normal ones.

Table 1. DTW / K-Means ($k = 1$).

	block 1	block 2	block 3	block 4	block 5
block 1	0.00%	18.21%	12.39%	12.76%	9.58%
block 2	8.68%	0.00%	9.89%	4.30%	5.90%
block 3	20.27%	30.65%	0.00%	18.04%	19.45%
block 4	9.35%	19.36%	11.34%	0.00%	8.16%
block 5	8.68%	16.92%	13.36%	9.50%	0.00%
-	min: 0.0, mean: 10.7, max: 30.6				
attack 1	34.25%	34.14%	34.59%	34.99%	34.99%
attack 2	42.26%	42.18%	42.39%	43.01%	43.05%
attack 3	43.52%	44.28%	43.74%	44.30%	43.85%
attack 4	77.65%	78.07%	74.43%	77.40%	77.55%
-	min: 34.1, mean: 49.5, max: 78.0				

Based on such tables, we can observe the difference among the novelty indexes in different classes of attack sessions: In the attack session 1 (class A) we observe the smallest amount of novelty, due to the fact that the vulnerability was exploited (i.e. the attacker gained administrative access), but no harmful action was performed. In sessions 2 and 3 (class B),

in which attack actions were executed after access was granted, we observe a large amount of novelty. However, the greatest amount of novelty at all is observed in session 4 (class C), in which a new attack was triggered from the victim host, leaving a large amount of attack traces behind.

Table 2. DTW / K-Means ($k = 16$).

	block 1	block 2	block 3	block 4	block 5
block 1	0.00%	19.37%	14.01%	13.83%	10.41%
block 2	9.51%	0.00%	10.86%	4.87%	6.60%
block 3	21.43%	32.30%	0.00%	19.16%	20.24%
block 4	10.33%	20.92%	13.00%	0.00%	8.92%
block 5	9.74%	18.47%	14.64%	10.66%	0.00%
-	min: 0.0, mean: 11.6, max: 32.3				
attack 1	38.32%	37.97%	38.71%	39.10%	39.24%
attack 2	46.26%	46.00%	46.20%	47.00%	47.11%
attack 3	47.51%	48.41%	47.75%	48.31%	48.01%
attack 4	80.55%	80.53%	78.98%	80.49%	80.65%
-	min: 38.0, mean: 53.4, max: 80.6				

Comparing the results of Tables 1 and 2, we see the increment of the quality of the system characterization when the proposed approach is applied. When $k = 1$, the series observations represent only the system call type, while when $k = 16$, we aggregate more information to the series, since each system call of the same type can now assume at most 16 different representations in the time series, and the series represent not only the system call type, but also parameters of calls.

We use two measurements to assess how the novelty detection techniques are capable of distinguishing the normal sessions from attacks: the area under the ROC curve, presented later in the paper, and the difference in between the novelty level observed at normal and attack situations. This latter is given by the difference between the arithmetic mean of the novelty level on attack situations, and the arithmetic mean of the novelty level on normal situations. Throughout the rest of this paper, we refer to this index as "mean difference".

For each clustering algorithm, we analyze the increase in the mean difference index in two situations: when we have the highest quadratic error index (which happens when all observations are clustered in a single centroid) and when we have the lowest quadratic error index of all executed experiments (which is observed when the number of centroids is increased). Table 3 shows the increase in the mean difference (considering the attacks under classes A, B and C). In the first section of the table we present results of testing with the attacks under class A, where no harmful action was performed after the root access was granted. On the second session, we present the results of attacks under class B, where com-

mon attack commands were executed, while the final section shows results of attacks under class C, in which a new attack was launched from the victim host. We observed that DTW was the only technique that did not present a significant change in the detection indexes after clustering: all other mean differences passed through a resampling test of significance [**?**] with a 90% confidence interval.

Table 3. Separation increase.

Class A	Adaptive LF	GWR	Simple LF	K-Means	Online KMLF
Entropy	12.30	6.98	14.59	17.34	15.35
Sliding Window	7.14	9.37	9.05	9.89	8.70
DTW	2.84	3.31	3.16	3.23	3.12
Class B					
Entropy	10.55	5.50	11.84	14.29	12.34
Sliding Window	7.81	9.77	9.21	10.60	9.04
DTW	2.95	3.28	3.07	3.15	3.25
Class C					
Entropy	−13.28	−34.20	−9.31	−12.68	−9.65
Sliding Window	13.70	13.75	14.04	11.46	14.23
DTW	1.19	2.10	1.79	2.34	1.98

For classes A and B, we observed a considerable increase in detection indexes for the Entropy and the Sliding Window techniques, showing that the added information helped to differentiate normal and attack situations. On the other hand, by analyzing section C, we noticed that the added information improved the differentiation for the Sliding Window technique, while it interfered on the differentiation of the Entropy technique.

4.3 Performance

Regarding execution time, all on-line clustering algorithms presented similar performance. Tests were executed on a 2-Core CPU computer at 2.8GHz, taking in average 4 minutes to process the dataset. On the other hand, the K-Means algorithm took 4 hours to process the dataset.

The performance of the detection algorithms is significantly different. Entropy is at a great degree the fastest of the evaluated techniques: it only depends on the observation of series state transitions, executing in linear time. The performance of the Sliding Window depends not only on the window length, but also on the amount of novelty found on a sequence. The Entropy processing time (i.e. the time to build a whole detection table, like Table 1) was around 2 seconds. The Sliding Window takes in average 9 minutes to process the dataset, but this processing time depends on the amount of information presented in the series: to evaluate the sequences generated by K-Means with $k = 1$ (i.e. with the smallest amount of information) the algorithm took 3 minutes, while the evaluation of the series generated with

$k = 16$ took 12 minutes. DTW takes around 2 hours to process the dataset.

4.4 Correlation Between Results

To conclude the validation of the proposed approach, we analyzed the correlation in between clustering indexes and the mean difference in between normal and attack situations. We plot the mean difference along its 90% confidence interval. Plots correspond to the Sliding Window novelty detection when applied to the dataset, considering the clustering with Greedy Leader-Follower and the single-sweep GWR approach (i.e. both adaptive algorithms).

Figure 4 present the values of mean difference and clustering indexes according to the number of centroids obtained from the series. We plot normalized clustering indexes, so that both clustering and novelty detection indexes can be observed on the same scale. A significant improvement is observed in the detection indexes when the number of centroids is increased. We also observed a strong correlation in between the increase in clustering quality and the capacity of differentiation in between attack and normal situations, showing that the system call parameters aggregated to the series bring up useful information, and that on-line clustering is a valid approach to the quantization of production systems.

Table 4. Mean Difference Correlation.

Q.E.	Adaptive LF	Online KMLF	Simple LF	K-Means	GWR
Entropy	0.90	0.98	0.99	0.81	0.22
Sliding Window	0.94	0.99	0.99	0.83	0.79
DTW	0.99	0.99	0.99	0.94	0.94
Silhouette					
Entropy	0.73	0.92	0.99	0.67	0.29
Sliding Window	0.83	0.96	0.97	0.67	0.51
DTW	0.83	0.95	0.97	0.84	0.76

Table 4 shows the correlation between the clustering indexes and the mean difference for all detection and clustering algorithms for both datasets. Except for eventual outliers, most algorithms present high correlation between clustering and detection indexes. Considering 14 degrees of freedom and $\alpha \leq 0.05$, the Student's t two-tailed critical value for Pearson's correlation is 0.49. Except of the Entropy with GWR experiments, all the observed correlation values are above this critical value.

Finally, we evaluated ROC curves obtained when using the novelty level to classify sessions as normal or under attack. The area under the ROC curve was used as indicator of classificator accuracy due to the fact that this index is not biased [19] when the number of observations is unbalanced (what happens for most anomaly detection datasets). To obtain the curve, we assumed that the normal and the attack novelty indexes generate two normal

distributions, and estimate their means and standard deviations based on the observed data. Figure 5 presents the true positive rate (TPR) by false positive rates (FPR), i.e. ROC curve, obtained when applying the adaptive clustering algorithms and use the Entropy as novelty index (for clarity, only most relevant curves are presented). For both experiments we noticed better differentiation (represented by curves closer to point $(0, 1)$ and area under curve closer to 1.0) when the number of centroids is increased (i.e. when we have better clustering quality, and hence a greater amount of information on the generated series).

For all experiments, when DTW or the Sliding Window is used as the novelty detection technique, the area under the curve varies between 0.979 and 1.0, due to the fact that those algorithms present higher differentiation capacity even for series with small amount of information. On the other hand, the area under the curve when considering the Entropy ranges from 0.59 to 1.0 (with an average of 0.96), making possible the analysis of the correlation between the clustering indexes and the area under the curve. Table 5 presents the correlation values for the on-line adaptive clustering algorithms. All experiments present high correlation indexes, which indicates that the main hypothesis of this work can also be observed for this differentiation index.

Table 5. AuC Correlation with entropy as novelty index.

	Adaptive LF	GWR
AuC x Q.E.	0.97	0.90
AuC x Silhouette	0.90	0.78

5 Conclusions

In this paper, we evaluated an approach for the characterization of unstructured application behavior and novelty detection aiming at performing intrusion detection. To allow the characterization of unstructured data, we make the quantization of the observed application behavior, reducing a complex stream of data to its essential aspects. This simplified structure is then used to generate time series where novelty detection is then performed. To be useful in a production environment, both clustering and novelty detection algorithms must work in a single-sweep fashion, where observations are verified once and then discarded, keeping in memory only a small amount of information, regarding the observed behavior. We have applied several on-line techniques to a system characterization dataset and compared the on-line techniques with baseline batch approaches, and we observed that those algorithms were capable of characterizing a working system with precision equivalent to batch approaches and also enough precision to distinguish normal and attack situations.

Since all clustering algorithms showed similar performance indexes, the choice of an algorithm may be based on how simple it is to employ it on given domain. The Simple

Leader-Follower and the On-line K-Means Leader-Follower algorithms heavily depend on the definition of an accurate value for parameter θ, which is typically defined through a trial and error approach, what is not an option on a real-world scenario. GWR and Greedy Leader-Follower, on the other hand, present adaptive approaches that are not so dependent on the parameter definition. Comparing the adaptive approaches, it is noticeable that Greedy Leader-Follower is easier to use due to two factors: first, it does not demand the definition of a point merging function, which in fact may not be available in certain domains. Second, it has less input parameters, and they are easier to estimate than the GWR ones.

Choosing a detection technique is not a simple task. The Entropy technique is very fast and memory-efficient, allowing a good characterization of a working system with very little computational resources. The Sliding Window, on the other hand, is more resource intensive and not so fast, but is capable of dealing with a much greater amount of information and can indeed characterize a system with greater precision. So, for any domain, if a good characterization in the clustering stage is feasible, the Entropy-based technique may be more useful as it is less resource intensive. Otherwise, the Sliding Window technique is more appropriate, as it is able to deal with series with limited or excessive amount of information.

Although we have used a distance function that is designed specifically for system calls, in order to have more precise indexes of clustering quality and novelty detection, we do not consider that the precision of this component is fundamental to build a working solution. As future work, we plan to evaluate the proposed techniques using distances that are not specific to the evaluated problem (such as Normalized Compression Distance [20]) that theoretically might be applicable to any problems. We also want to evaluate the technique with datasets collected from other types of applications (such as a 3-tier web application), without any pre-processing or data normalization. We also plan to evaluate the scalability of the technique by applying it to production-scale datasets.

References

[1] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128, 1996.

[2] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[3] Jamie Twycross and Uwe Aickelin. Information fusion in the immune system. *Information Fusion*, 11(1):35–44, 2010.

[4] Anil K Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice Hall, first edition, 1988.

[5] Markos Markou and Sameer Singh. Novelty detection: a review, part 2: neural network based approaches. *Signal Processing*, 83(12):2499–2521, 2003.

[6] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, 2002.

[7] Rui Xu and Don Wunsch. *Clustering (IEEE Press Series on Computational Intelligence)*. Wiley-IEEE Press, first edition, 2008.

[8] Moses Charikar, Chandra Chekuri, Tomï£¡s Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 626–635, 1997.

[9] Stephen Marsland, Jonathan Shapiro, and Ulrich Nehmzow. A self-organising network that grows when required. *Neural Networks*, 15(8):1041–1058, 2002.

[10] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[11] Peter Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53–65, 1987.

[12] Lucas Vendramin, Ricardo J. G. B. Campello, and Eduardo R. Hruschka. On the comparison of relative clustering validity criteria. In *SIAM International Conference on Data Mining*, pages 733–744, 2009.

[13] Richard Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.

[14] Cássio M. M. Pereira and Rodrigo Fernandes de Mello. Behavioral study of unix commands in a faulty environment. In *Proceedings of the 8th International Conference on Dependable, Autonomic and Secure Computing*, pages 1–6, 2009.

[15] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(1):379–423, 1948.

[16] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information System Security*, 3(4):262–294, 2000.

[17] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on security and Privacy*, pages 133–145, 1999.

[18] Christopher Kruegel, Darren Mutz, Fredrik Valeur, and Giovanni Vigna. On the detection of anomalous system call arguments. In *Proceedings of the 8th European Symposium on Research in Computer Security*, pages 326–343, 2003.

[19] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

[20] Rudi Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, 2005.

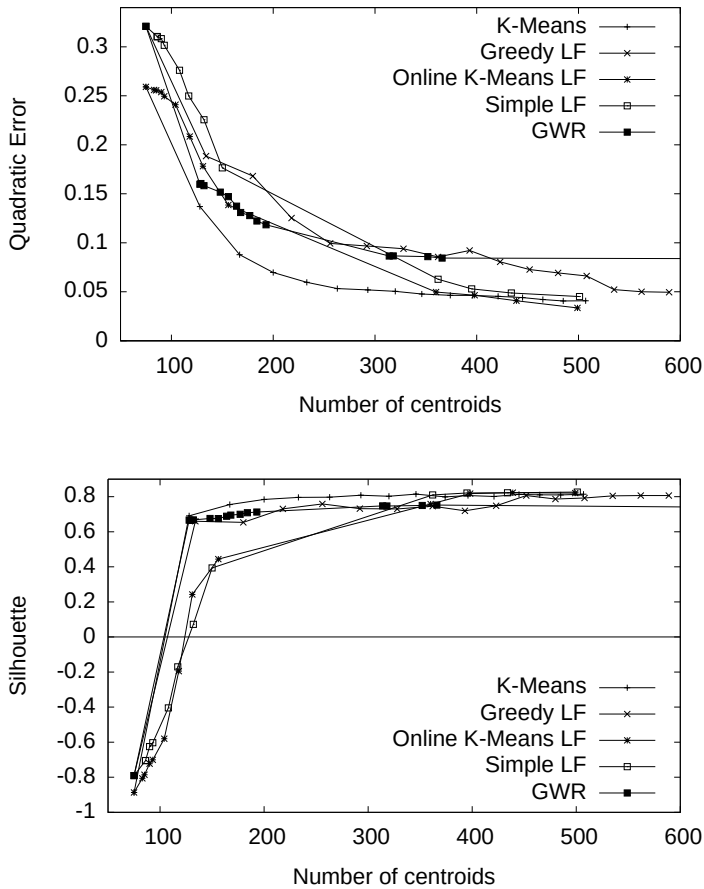
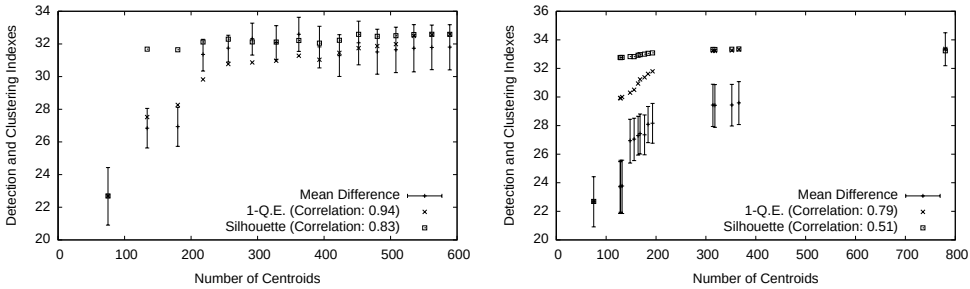
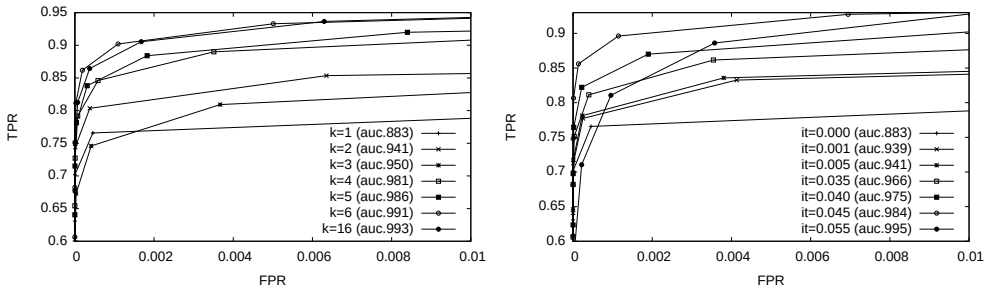


Figure 3. Clustering indexes.



(a) Adaptive LF (b) GWR

Figure 4. Correlation between clustering and differentiation indexes.



(a) Adaptive LF (b) GWR

Figure 5. ROC curves with Entropy as novelty index.