

Some experiments on activity outlier detection

(Discussion Paper)

Michele Ianni¹, Elio Masciari²

¹Università della Calabria, Italy

²Università Federico II, Napoli, Italy

Abstract

The rise of cyber crime observed in recent years calls for more efficient and effective data exploration and analysis tools. In this respect, the need to support advanced analytics on activity logs and real time data is driving data scientist' interest in designing and implementing scalable cyber security solutions. However, when data science algorithms are leveraged for huge amounts of data, their fully scalable deployment faces a number of technical challenges that grow with the complexity of the algorithms involved and the task to be tackled. Thus algorithms, that were originally designed for classical scenarios, need to be redesigned in order to be effectively used for cyber security purposes. In this paper, we explore these problems and then propose a solution which has proven to be very effective in identifying malicious activities.

Keywords

Cybersecurity, Data Compression, Activity Analysis, Cluster Analysis

1. Introduction

Monitoring activities is a crucial and often underused method in the task of documenting, assessing and preventing risk. Either if we talk about a large computer network or a single machine or even a small IoT device, we face the problem of handling a high volume of data that is generated continuously. This data could be related to network activity, software logs, user interaction, hardware monitoring to cite a few. By carefully analyzing this endless stream of information, it could be possible to thwart many kinds of cyberattacks, even before they happen [1].

Cybercrime is on the rise, annual damage from cyberattacks is set to hit 6 trillion dollar in 2021 [2], this is twice the amount of annual damage from cybercrime recorded in 2016. Many attacks follow well known patterns and many malicious software are just a variant of existing menaces, however, even the simplest form of cyberattacks are still dangerous nowadays. It is important to notice that many attacks could be detected in the very early stages, or even prevented at all, if all the information available could be analyzed in a fast and reliable way [3]. In this respect, activity monitoring is a crucial component of comprehensive cybersecurity. The technologies in the field of Big Data analysis and the advances of computing capabilities led to


SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy

✉ michele.ianni@univr.it (M. Ianni); elio.masciari@unina.it (E. Masciari)

ORCID 0000-0003-0562-7462 (M. Ianni)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

new possibilities in the difficult task of extracting sensitive information from huge amounts of data [4]. Moreover, activity data has an inestimable value in cybersecurity as the information extracted can easily be leveraged in order to better frame the legitimate actions performed thus easily recognizing and blocking unwanted interactions.

Nowadays, especially in business scenarios, the attackers could be insiders, even employees sometimes, who (consciously or not) could endanger the sensitivity of data, or damage the software/hardware architecture by opening holes that can be exploited by attackers [5]. Mis-configured services, unpatched binaries, unknown vulnerabilities are all issues who constantly jeopardize the security of our computers, leaving big holes open which can be used by attackers to easily infiltrate. By analyzing logs and monitoring activities in real time, many of these holes could be detected before an unauthorized user could spread your valuable information [6].

How Is Activity Monitoring Useful In The Event Of An Attack? If a cyberattack occurs, activity logs provide a roadmap to the origin of the attack and what agents were directly involved. Each log can provide further details about where the cyberattackers may be going next, how to prevent further attacks and what can be done to secure your system after the threat has been eliminated. It is important to notice that, in the majority of the cases, the domain experts are able to classify malicious actions and safe ones. This is usually a straightforward task that is performed in several different types of security systems, from intrusion detection systems to blacklist based access control. Unfortunately, the task of identifying and marking malicious actions is not always easy. As a matter of fact, even if we could enumerate all malicious actions, we may find that many malicious activities are the result of the execution of several actions that, when analyzed individually, are usually considered safe. This allows to understand why simple blacklists are usually not enough to keep a system safe. System administrators are overwhelmed by a continuous stream of logs that is very difficult to tackle. It is worth noticing that, in managing this huge amount of data, it is likely to miss small amounts of relevant actions, that could be related to an attack. Indeed, the actions related to an attack represent an almost invisible fraction, in terms of size, hidden in endless logs as a needle in a haystack. Thus, it is straightforward to note that the analysis of the activity logs became an infeasible task to be performed manually, on the contrary it is mandatory to use anomaly detection techniques in order to early detect any possible threat.

In this paper, we leverage the prime numbers based encoding scheme described in [7] in order to efficiently and effectively identify suspicious activities by means of a proper outlier detection strategy. More in detail, our contribution can be summarized as follows:

- **scout**: a hierarchical approach for quick identification of outlying activities;
- A deep experimental evaluation of the proposed approach that confirmed the validity of our proposal.

2. Computing Outliers on Activity Logs

In order to identify outlying activities after they are encoded as explained in [7], we leverage a fast hierarchical approach, that combines the benefits of the divisive and agglomerative approaches. Fig. 1 illustrates the operations performed by our algorithm **scout**. The input

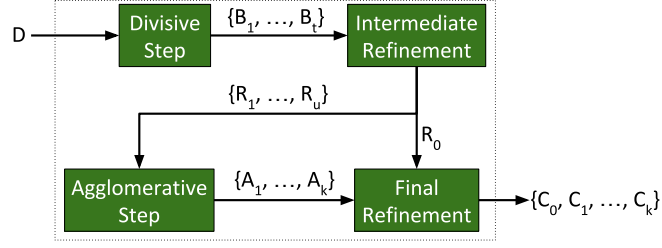


Figure 1: The four Steps of the SCOUT Algorithm

dataset D is first partitioned into a set of rectangular blocks $\{B_1, \dots, B_t\}$ by means of a top-down divisive procedure. Those blocks are then refined in the next step, whose result is a new partitioning of the points of D , $\{R_0, R_1, \dots, R_u\}$, where R_0 contains points that are classified as outliers, and each of the remaining u sets represents a cluster. These u clusters are processed in the third step, through a bottom-up agglomerative procedure, whose result is a new partitioning of the points in $R_1 \cup \dots \cup R_u$ into the sets A_1, \dots, A_k , where $k \leq u$, obtained by possibly merging groups of clusters in $\{R_1 \cup \dots \cup R_u\}$ into one only cluster of $\{A_1, \dots, A_k\}$. Finally, the new clusters in output from the agglomerative step and the outliers R_0 in output from the previous, are refined, thus obtaining the final set of well-rounded clusters, $\{C_1, \dots, C_k\}$, and the set of outliers C_0 . In our running example of Fig. 2, we show how the successive steps of SCOUT applied to the two-dimensional data distribution of Fig. 2(a), produce the results shown in Fig. 2(b–e). Thus, Fig. 2(b), (c), (d), and (e) show the results produced, respectively, by the steps of Fig. 1, which are described in detail in the following.

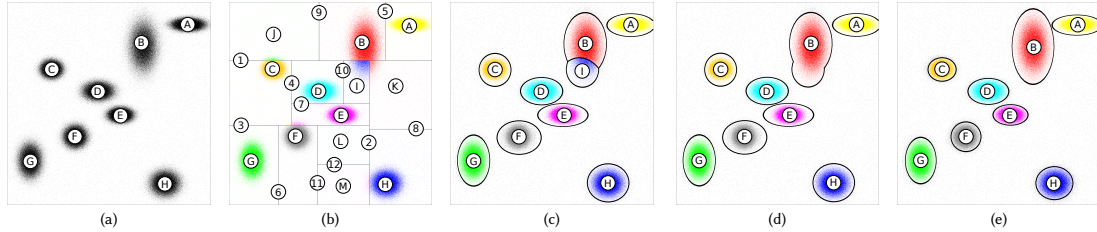


Figure 2: A two-dimensional data set with 8 clusters (a), its partitioning after the divisive step (b), the intermediate refinement (c), the agglomerative step (d), and the final refinement (e).

The divisive step of SCOUT performs a top-down partitioning of the data set to isolate blocks whose points are as much as possible close to each other: this is equivalent to minimizing the $WCSS$ of the blocks, an objective also pursued by the vast majority of algorithms tailored for outlier detection. As the clustering algorithm at the basis of the outlier detection strategy have been described in details in [8, 9] we do not report here the complete description except a brief overview of the approach focusing on the goal of this paper, i.e., the outlier detection phase. Indeed, the divisive phase described above partition the overall space into (A) several blocks containing clusters, and (B) zero or more blocks that only contain **outliers** and noise points. We also see that blocks of type A will only contain a single cluster since blocks containing several clusters were split according to the “when in doubt split” philosophy used in the divisive

phase. To better understand this approach, we report the details of the refinement step in order to better explain the details of the outlier detection strategy. More in detail, *scout* seeks to realize the objectives of (I) identifying the blocks that contain only outliers, and (II) generating well-rounded clusters for the remaining blocks. Task (I) is performed by using the observation that (a) the density of blocks containing only outliers is low, and (b) the density of such blocks is rather uniform because of the random nature of noise and outliers.

Algorithm 1: The algorithm for identifying outlier blocks

```

1 Procedure identifyOutlierBlocks( $B$ )
   Input:  $B = \{B_1, \dots, B_t\}$ : a set of  $t$  blocks
   Output:  $o$ : an array of  $t$  boolean values, where  $o_i = true \Leftrightarrow B_i$  is classified as outlier block
   begin
2      $o \leftarrow$  an array  $[o_1, \dots, o_t]$  of  $t$  boolean;
3      $den \leftarrow$  an array  $[d_1, \dots, d_t]$  containing the densities of blocks in  $B$ ;
4      $v \leftarrow$  an array  $[v_1, \dots, v_{t-1}]$  containing  $t-1$  numbers;
5     sort( $den$ );
6     for  $i \leftarrow 1$  to  $t - 1$  do
7        $v[i] \leftarrow den[i + 1]/den[i]$ ;
8     end
9      $vt \leftarrow$  average( $v$ ) + standardDeviation( $v$ );  $vi \leftarrow 1$ ;
10    while  $vi < t \wedge v[vi] \leq vt$  do
11       $vi \leftarrow vi + 1$ ;
12    end
13     $dt \leftarrow den[vi]$ ; // the maximum density an outlier block may have
14    for  $i \leftarrow 1$  to  $t$  do
15       $o[i] \leftarrow density(B_i) \leq dt \wedge \text{aroundCentroidDensity}(B_i) < 2 \times density(B_i)$ ;
16    end
17    return  $o$ ;
18  end
19

```

The pseudo-code is reported in Algorithm 1. *scout* checks condition (a) first by sorting the densities of blocks and detecting discontinuity in density between blocks that are contiguous in that ordering. That is, if den is an array containing the density values of the t blocks yielded by the divisive step, sorted in ascending order, we compute an array v with $t - 1$ values, such that $v[i] = den[i + 1]/den[i]$. Then, if vi outlier-blocks are present, the first $vi - 1$ values of v will be approximately equal to 1, while $v[vi]$ will be ‘quite large’. Through a number of experiments, extensive empirical evaluation shows that a good threshold for value of v to be considered ‘quite large’ is provided by average value of v plus its standard deviation. Thus, with vi the smallest value index for which the v -value exceeds this threshold, we classify all blocks with density above $dt = den[vi]$ as cluster blocks. Then, we test the other blocks using condition (b), i.e., we test the blocks that are suspected to be outlier blocks because of their low density. This additional verification step is needed because large density differences between cluster blocks cannot be always excluded (e.g., because of large differences in the volume of those blocks). Therefore, a small hypercube is taken around the centroid of each block being checked for condition (b), as depicted in Figure 3, and if the density in the hypercube is less than twice the density of the whole block, this is classified as an outlier block (Fig. 3(a)); otherwise it is classified as a cluster block (Fig. 3(b)). The size of the hypercube used for this check is such that, for each dimension, the edge of the hypercube is 1/10 of that of the whole block on the same dimension. The rationale of this the criterion can be understood by considering that for an

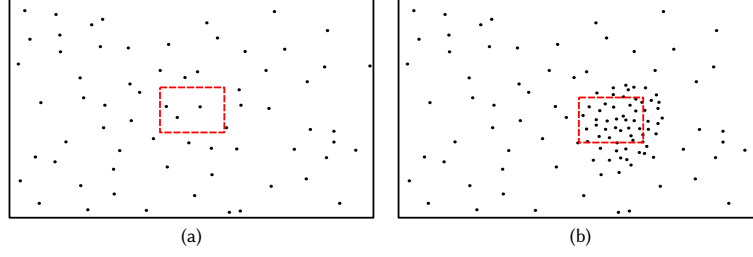


Figure 3: A block containing only outliers (a) and a block containing a cluster (b).

outlier-block, where data are randomly distributed, it is very unlikely that the density around centroid is twice the global density while in a cluster-block, the density around centroid is very likely to be twice the global density. For instance, these tests applied to our running example in Fig. 2(b) have classified blocks “J”, “K”, “L” and “M” as outlier-blocks.

Having thus identified all the cluster blocks, we can now proceed with task (II) and materialize the centroids of their clusters and assign each point to the cluster with the nearest centroid. To this end, we use a weighted function of the distance between points on each dimension. Specifically, we measure the distance of a point $\mathbf{p} = (p_1, \dots, p_d)$ from a cluster-blocks C with centroid (c_1, \dots, c_d) as

$$dist^*(\mathbf{p}, C) = \sum_{i=1}^d \left(\frac{\|p_i - c_i\|}{r_i} \right)^2$$

where r_i is the radius of the cluster along the i -th dimension, which is estimated by the usual $3 \times \sigma$ criterion [10] used to separate core points from outliers:

$$r_i = 3 \cdot \sqrt{\frac{WCSS_i(C)}{n}}$$

where n is the number of points inside C and $WCSS_i(C)$ is the variance of the i -th coordinates of the points in C . Observe that $dist^*(\mathbf{p}, C) = 1$ defines an ellipsoid whose center is the centroid of C and whose radius on the i -th dimension is r_i . Therefore, we can now classify each point in the sample set as either belonging to a particular cluster or as an outlier by assigning each point \mathbf{p} to the cluster C which minimizes $dist(\mathbf{p}, C)$, provided that this minimum distance is less than or equal to 1; otherwise \mathbf{p} is classified as outlier. The final output is thus a quite simple and explainable set of points that are marked as outliers.

3. Experimental Evaluation

In this section, we will describe the experimental assessment for our framework. First, we introduce the pre-elaboration steps that are needed on the dataset before running our tests as the activity logs usually contains a huge amount of information that could be not partitioned in sessions or single user activities. More in detail, when dealing with transactional data, a tuple is a collection of features, an activity record instead is an ordered set (i.e., a sequence) of

time-stamped actions. Moreover, log data are usually recorded in a variety of different formats, and they can be built from several domains. As we deal with activity logs composed of different types of actions we define a mapping function from the domains of actions to logs.

Definition 3.1. Let ACT be a set of actions contained in an activity logs, a function $L : O \rightarrow ACT$ mapping each action to its activity log (or set of activity logs) is called a *labeling* function.

We point out here, that due to privacy reasons, real datasets are usually anonymized, nevertheless, in the experimental section we performed a proper labeling of activities in order to obtain a meaningful experimental assessment for our approaches. More in detail, we ran our experiments on several dataset whose naming convention are different for each action, thus, we omit here to describe the labeling function we used as they are quite straightforward (by using name permutation or punctuation) but we mention here their existence to clarify that a proper mapping is always computed as pre-elaboration step especially when same action could have different name in different sessions in order to have a fair comparison.

3.1. Results

For the experimental evaluation we used two different datasets.

The first dataset source is the widely used KDD Cup 1999 Data [11] which contains more than seven million connection records. A connection is a sequence of TCP packets starting and ending within a time interval during which, data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes. Attacks fall into four main categories: i) *DOS*: denial-of-service, e.g. syn flood; ii) *R2L*: unauthorized access from a remote machine, e.g. guessing password; iii) *U2R*: unauthorized access to local superuser (root) privileges, e.g., various “buffer overflow” attacks; iv) *probing*: surveillance and other probing, e.g., port scanning. As regards the second dataset, referred from now on as “synthetic”, in order to perform fine tuning tests, we developed a logging system based on the `strace` Linux command that captures the system calls invoked by a malicious software we created ad-hoc. The generated log contains the timestamp, the user performing the call, the name of the call, and its parameters. We further enriched the information contained in the logs by running our logging system on many different applications, using different users, so that the results contains information related both to normal usage of the operating system and to malicious activity. It is important to notice that each dataset described above contains labels that are used to mark malicious actions. However, since we are dealing with activities which we defined as a sequence of several atomic actions, we consider an activity as a malevolent one if it is a sequence of actions containing at least one malicious action. The first task in our experimental evaluation is, then, identifying activities in our datasets. This task can be done using different metrics and considerations, and, in most of the cases, is strictly related to the specific log we are working on. The actions included in a given activity are, in fact, determined by means of multiple factors like the user performing the action, the timestamp, the relationship with a software session, a network connection, the type of the action performed and so on. More in detail, we built activities by specifying a fixed length and then (eventually) padding them as explained in previous section. We compared the performances on well-established

<i>Measure</i>	<i>Derivations</i>	<i>KDD Cup</i>	<i>Synthetic</i>
Sensitivity	$\frac{TP}{TP+FN}$	0.9700	0.9920
Specificity	$\frac{TN}{FP+TN}$	0.8829	0.9920
Precision	$\frac{TP}{TP+FP}$	0.9868	0.9920
Negative Predictive Value	$\frac{TN}{TN+FN}$	0.7656	0.9920
False Positive Rate	$\frac{FP}{FP+TN}$	0.1171	0.0080
False Discovery Rate	$\frac{FP}{FP+TP}$	0.0132	0.0080
False Negative Rate	$\frac{FN}{FN+TP}$	0.0300	0.0080
Accuracy	$\frac{TP+TN}{P+N}$	0.9613	0.9920
F1 Score	$\frac{2TP}{2TP+FP+FN}$	0.9783	0.9920
Matthews Correlation Coefficient	$\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)*(TP+FN)*(TN+FP)*(TN+FN)}}$	0.8011	0.9840

Table 1: KDD Cup and Synthetic datasets

evaluation described in table 1 where TP , FP , TN , FN respectively mean True Positive, False Positive, True Negative and False Negative.

In Figure 1, we report the results obtained for the KDD and Synthetic datasets. The performances are quite interesting due to the peculiar features of our approach, especially when observing that the percentage of threats that are not identified correctly (and the percentage of normal activities marked as malicious as well) is negligible.

4. Conclusion and Future Work

In this paper we discussed SCOUT, an algorithm for clustering based outlier detection for activity logs. Our goal is to overcome the limitation of classical approaches for outlier detection that do not apply well to activity logs as each activity is a sequence of actions and the definition of a proper metric for evaluating anomalous actions could be quite difficult. In this respect, we leverage a simple yet effective encoding that allow us to easily represent complex activities as real numbers. After the encoding phase, we perform an effective outlier detection based on cluster analysis. We evaluated our algorithm on several datasets by performing an accurate pre-elaboration step in order to make them more effective. The results obtained show a very good effectiveness of our strategy on all the datasets being tested thus confirming the soundness of our approach. As a future work, we plan to test other encoding strategies as well as more refined outlier functions in order to further improve our (already satisfactory) performances.

References

- [1] J. Babbin, Security log management: identifying patterns in the chaos, Elsevier, 2006.
- [2] S. Morgan, Cybercrime to cost the world \$10.5 trillion annually by 2025, 2020. URL: <https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/>.

- [3] J. Jang-Jaccard, S. Nepal, A survey of emerging threats in cybersecurity, *Journal of Computer and System Sciences* 80 (2014) 973–993.
- [4] P. Zikopoulos, C. Eaton, et al., *Understanding big data: Analytics for enterprise class hadoop and streaming data*, McGraw-Hill Osborne Media, 2011.
- [5] E. E. Schultz, A framework for understanding and predicting insider attacks, *Computers & Security* 21 (2002) 526–531.
- [6] K. Kent, M. Souppaya, *Guide to computer security log management*, NIST special publication 92 (2006) 1–72.
- [7] M. Ianni, E. Masciari, A compact encoding of security logs for high performance activity detection, in: *29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2021, Valladolid, Spain, March 10-12, 2021, IEEE, 2021*, pp. 240–244. URL: <https://doi.org/10.1109/PDP52278.2021.00045>.
- [8] M. Ianni, E. Masciari, G. M. Mazzeo, M. Mezzanzanica, C. Zaniolo, Fast and effective big data exploration by clustering, *Future Generation Computer Systems* 102 (2020) 84–94.
- [9] M. Ianni, E. Masciari, G. M. Mazzeo, C. Zaniolo, Clustering goes big: Clubs-p, an algorithm for unsupervised clustering around centroids tailored for big data applications, in: *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), IEEE, 2018*, pp. 558–561.
- [10] T. Zhang, R. Ramakrishnan, M. Livny, Birch: An efficient data clustering method for very large databases, in: *SIGMOD, 1996*, pp. 103–114.
- [11] S. Hettich, Kdd cup 1999 data, The UCI KDD Archive (1999).