Heterogeneous Security Events Prioritization Using Auto-encoders

Alexandre Dey[1,2,3(✉)], Eric Totel[1], and Sylvain Navers[3]

[1] IMT-Atlantique, Nantes, France
eric.totel@imt-atlantique.fr
[2] Inria, Univ. Rennes, IRISA, Rennes, France
[3] Airbus CyberSecurity, Elancourt, France
{alexandre.dey,sylvain.navers}@airbus.com

Abstract. In a large monitored information system, analysts are confronted with a huge number of heterogeneous events or alerts produced by audit mechanisms or Intrusion Detection Systems. Even though they can use SIEM software to collect and analyse these events (In this paper we call events all events or alerts produced by the monitoring processes), detecting previously unknown threats is tedious. Event prioritization tools can help the analyst focus on potentially anomalous events. To compute a measure of priority among events, we propose in this paper to define the notion of an anomaly score for each attribute of the analyzed events and a method for regrouping events in clusters to reduce the number of alerts the analysts have to qualify. The anomaly score is computed using neural networks (i.e., auto-encoders) trained on a normal dataset of events, and then used to provide the analyst with the information of the difference between normal learned events and the events actually produced by the monitoring system. Additionally, the auto-encoders also provide a way to regroup similar events via clustering.

Keywords: Heterogeneous logs · Anomaly detection · Anomaly score · Cybersecurity · Intrusion detection · Machine learning

1 Introduction

Security monitoring of information systems requires to log events happening during the execution of processes at system level, the exchange of data via the network or the application warnings. In addition to event logging, Intrusion Detection Systems can produce alerts that are likely to be the consequence of an attack. Due to the huge number of events produced, even if a monitoring strategy has been clearly defined, it is difficult for the analysts to detect what are important events from the security point view, i.e., what are the events that are symptomatic of an intrusion inside the system.

The current practices consist in collecting all security events in a SIEM (Security Information and Event Management) solution. This solution is able to correlate information included in multiple events in order to recognize known attack

J. Garcia-Alfaro et al. (Eds.): CRiSIS 2020, LNCS 12528, pp. 164–180, 2021.
https://doi.org/10.1007/978-3-030-68887-5_10

patterns. Despite the definition of highly accurate correlation processes [14], this treatment still requires to manually write static correlation rules. Thus, the effectiveness of the detection relies on the ability of the analysts to write a complete set of correct correlation rules for known attacks. Furthermore, this set of rules should be updated continuously to take into account the newly discovered threats. As a consequence this tremendous task is clearly insufficient to emphasize all attack steps, and a lot of anomalous events stay hidden to the analyst.

During the threat hunting process, analysts rely on prioritization tools to highlight the most anomalous events and identify misbehaving entities in the system. If necessary, a more thorough forensic analysis of these entities can be performed. After this analysis, they should be able to produce a set of Indicators of Compromise (IoC) and eventual correlation rules. As a way of prioritizing events, in this paper, we propose an approach which associates an anomaly score to each attribute of an event[1]. These per attribute scores are then combined to provide a global anomaly score to the event. The higher this score is, the lower the probability of it being a consequence of a normal behavior is. This approach relies on the use of Artificial Intelligence mechanisms, more specifically, neural networks auto-encoders. The originality of the approach is that the computation is applied to any type of events (i.e., network, system and application events). While other related methods require complex feature engineering to transform attributes into compliant inputs for the chosen algorithms (e.g., for strings, choice between feature hashing, one-hot encoding, TF-IDF, etc.), our method only requires analysts to identify events attributes as being a numerical, categorical or string variable. This makes it easier to adapt to new category of security event. A major contribution of our approach is the introduction of a way for auto-encoders to provide a cluster identifier to each event. The identifier is used to easily and accurately regroup similar events. This clustering lowers the volume of redundant information presented to analysts which lead to almost three orders of magnitude reduction for our test dataset. The main advantage of our method is the possibility to rapidly adapt it to new security event sources, providing as output a cluster identifier and an anomaly score to the analyzed events. This paper is organized as follows: Sect. 2 presents the state of the art in anomaly detection using Artificial Intelligence techniques. Section 3 explains how the anomaly score is computed. In Sect. 4 we describe how the approach is implemented. Finally Sect. 5 presents the results obtained on a data set produced using an environment of heterogeneous Operating Systems on an internal network.

[1] Attributes are the fields of an event. Connection duration, source IP address, number of bytes received are examples of attributes for a network event.

2 State of the Art in AI Applied to Security Monitoring

Our approach permits to categorize attributes of events as being normal or abnormal. A lot of work uses Artificial Intelligence approaches to attain a similar objective (i.e., detecting anomalies), mainly machine learning techniques. Kriegel et al. [3] computes the anomaly score based on the distance with the nearest neighbours, with a high distance to the other points indicating a potential anomaly. Pang et al. [19] proposed a nearest neighbours based method, that scales to larger datasets (several millions of events) by computing the pairwise distance between random samples of point instead of the whole dataset. Ester et al. [9] proposed DBSCAN, an approach that identifies high density of points as clusters and classify points inside low density region as anomalies. However, these approaches are sensible to a high dimensional data (the "curse of dimensionality" described by Bellman et al. [2]). As a consequence Kriegel et al. [13] proposed a work that scales to large number of attributes in data. All the methods mentioned above rely on a notion of distance that needs to be defined specifically for the problem at hand, which can prove difficult, especially for complex data structures (e.g., the distance between two strings, two events with heterogeneous attributes types, etc.)

A variant of Principal Component Analysis has been also used by Pascoal et al. [20] to propose an approach that is robust to noise in the training dataset (e.g., a few attack traces in the normal data). Scholkopf et al. [23] proposed one of the most used algorithm for anomaly detection by training SVM (Support Vector Machines). Data Mining techniques have been used by He et al. [11] to measure the level of anomaly of a transaction. This type of approach have been extended by Akoglu et al. [1] to limit the number of frequent pattern used to compute the anomaly score. Pattern mining algorithm requires categorical data as input, and therefore a suitable transformation of the input data should be found for numerical data.

The use of Bayesian Networks [22] has been tested by Wong et al. [27] to perform anomaly detection. This type of approach permits also to diagnose and explain a detected anomaly. However, Bayesian methods requires to identify the most likely probability distribution for the events, which can be challenging.

The algorithm Isolation Forest proposed by Liu et al. [15] was applied to security by Ding et al. [7]. This permits to classify quickly the abnormal activities. This technique does not require any kind of normalization on numerical variables, but it requires categorical values to be transformed into numerical values and cannot handle text values without specific transformation methods.

Similarly to our approach Hawkins et al. [10] propose a method based on neural networks to compute anomaly score. This approach is called Replicator Neural Networks (RNN). With the rise of Deep Learning and more specifically Deep Neural Networks (DNN), RNN have regain interest in the form of deep auto-encoders and a robust variant of the algorithm has been proposed by Zhou et al. [28]. Mirsky et al. [18] relies on an ensemble of auto-encoder to improve the robustness and accuracy. Due to recent advancements in deep learning, auto-encoders can be adapted to various kind of data (e.g., text, time-series, images,

categorical, numerical, etc.). However, such a network is computationally intensive to train and is best suited for high volume of training data. Veeramachaneni et al. propose an active learning based approach for large scale security monitoring [26]. The authors combine a Principal Component Analysis approach, auto-encoders and a distance-based approach for anomaly detection, but they still require complex feature selection and transformation for each event sources. In [8], a deep learning method is used to spot anomalous patterns inside log files. However, the considered threat model is mainly focused on Denial of Service and workflow interruption, and the method has therefore not been assessed on broader range of hostile behaviors.

Shen et al. [24] and Liu et al. [16] propose methods that rely on embedding of security-related information, like our method. The former's objective is to model the evolution of exploitation methodology for known vulnerability, and is therefore more related to cyber threat intelligence than security monitoring. The latter aims at detecting complete attack (i.e., not anomalous events) and relies on complex sets of rules to build graphs, which is hard to adapt to new types of security events and threat models.

Our approach draws inspiration from state-of-the art deep learning techniques to take as input numerical (e.g., connection duration, file size, etc.), categorical (e.g., port number, user identifier, hostname, etc.) and string (e.g., a command and its arguments) attributes. Doing so permits the use of simple and generic methodology for input transformation. In addition, we exploit the latent representation of the auto-encoder in a novel way to provide clustering capabilities. This is used to regroup similar event and lower the volume of information that is presented to the analysts.

3 Computing Anomaly Score on Heterogeneous Events

3.1 Basics on Neural Networks Auto-encoders for Anomaly Detection

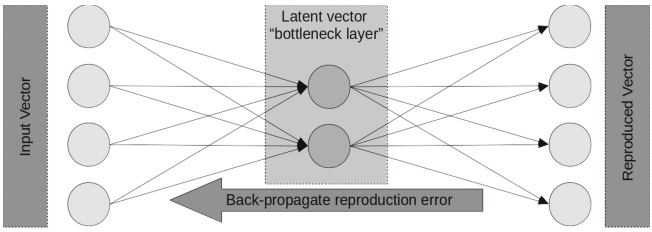

Fig. 1. Overview of an autoencoder

Auto-encoders (Fig. 1) are by definition a particular structure of neural networks that are trained in an unsupervised way (i.e., without the expected output value in the training dataset) and composed of an encoder, which maps input vectors to a low dimension representation (also called latent space), and a decoder which tries to reconstruct the original input vector from the latent space.

The input vectors are vectors containing numerical values (integers, floats or vectors of integers or floats). For anomaly scoring, the auto-encoder is first trained on normal data to compress and decompress the input vectors with as little loss of information as possible (i.e., an approximation of the identity function). Then, an inference phase that takes as inputs events produced during the monitoring process, to which we apply the auto-encoders estimated identity function. As output we obtain a result that can slightly differ from the input. We name this difference "reproduction error", and we use this difference to estimate the deviation of the input from learnt normal known inputs. More specifically, the auto-encoder is biased towards normal events, and therefore, the reproduction error is higher for anomalous events.

During the security monitoring process, an information system produces a huge number of heterogeneous events. These events can be for example extracted from the operating system (e.g., system calls), from network (e.g., connections, protocols, network IDS alerts, etc.), or from specific monitored applications (e.g., web server requests logs). Moreover, different types of operating systems can be used (e.g., Windows and Linux systems), generating different formats of events. Also, an event that is the consequence of a normal behaviour on one system, might be a sign of an intruder in another one. Our objective is to create normal behaviour models from all these heterogeneous events in order to compute, for each event, an anomaly score, and do so with a minimal configuration step.

Due to their ability to handle heterogeneous attributes, we chose to adapt neural network auto-encoders to compute an anomaly score (Sect. 3.5) for security events. We also exploit the latent space of these auto-encoders to provide a cluster identifier to each event, later used to regroup similar events in clusters (Sect. 3.4). In this section, the structure of the auto-encoders is provided (Sect. 3.3), as well as the transformations that are applied on the events before being processed by the auto-encoder (Sect. 3.2).

3.2 Input Transformation

As explained in Subsect. 3.1, the auto-encoder takes as input a vector of numerical values. The goal of the input transformation process is to transform the vector containing the value of the attributes of an event into a vector suitable as input for the auto-encoder. This transformation process is presented in Algorithm 1.

There are three possible types of variables that can be taken as input to our auto-encoders. The first type, like any machine learning algorithm, is numerical variables. The second one, the categorical variables, are variables whose values belong to a finite set and cannot be mathematically ordered (e.g., username 'Bob" is neither superior or inferior to username "Alice"). Finally, raw strings are considered as sequences of categorical values.

Normalizing Numerical Attributes. For numerical variables (integers and floats) taking raw values as input is not the most efficient way of learning the normal distribution of numerical attributes [12]. This paper proposes as a remediation to normalize the numerical values from a set of floats into a reduced

Algorithm 1. Event Attribute Vector Transformation

```
function TransformCategory(attribute)
    for all pattern in KnownPatterns do
        if matchPattern(pattern, attribute) then
            attribute ← pattern
            break
        end if
    end for
    if attribute in KnownCategories then
        return value ← KnownCategories[attribute]
    else
        return value ← KnownCategories[DefaultValue]
    end if
    return value
end function
function Transform(event)
    Vector ← ∅
    for all SelectedAttribute in event do
        switch SelectedAttribute.type do
            case categorical
                Vector[i] ← TransformCategory(SelectedAttribute)
            case string
                Vector[i] ← StringToIntegerArray(SelectedAttribute)
            case number
                Vector[i] ← ScaleValue(SelectedAttribute)
    end for
    return Vector
end function
```

interval (e.g., the set $[0, 1]$). This transformation is produced by the **ScaleValue** function in Algorithm 1.

In the context of anomaly detection, outliers with extreme values can reside in normal data. To limit their impact on the normalization process, we find the 90^{th} percentile Q_{90} inside the training dataset (i.e., 90% of the normal values are below Q_{90}). The transformation then consists in dividing the input by Q_{90}. In case the value of the attribute grows exponentially, as suggested by Kaastra et al. [12], the result of the transformation will be the logarithm of the initial value divided by the logarithm of Q_{90}.

Normalizing Categorical Attributes. To handle categorical variables in the auto-encoder, we draw inspiration from *word2vec* [17]. The goal of this technique is to map each word in a continuous vector space (i.e., vectors of floats) based on its context (other words appearing in the same sentences). This mapping is generally called an embedding. This permits to treat natural language, such as the recognition of semantics of words in sentences. We use the following analogy: an event is a sentence and its attributes are the words composing it.

The neural network will optimize the embedding function based on the other attributes of the given event, that will represent the context of the transformed attribute. This context allows us to determine if a category of an attribute is normal in a given context.

In practice, a categorical embedding layer of a neural network takes as input a category identifier (i.e., an integer) that represents a vector filled with as much

0 as the total number of categories, except for a 1 at the corresponding identifier. When the total number of categories is large, a proportionally large number of parameters needs to be optimized. To reduce the induced computational complexity, we propose the use of regular expressions (regex) to map every string matching the same pattern to the same category identifier. For example, if we consider all HTML files in a web server repository as being of the same category, we can map them to the same identifier using the regular expression **.html*. However, as with any security tools relying on regex, the regex should be carefully chosen to prevent an attacker from bypassing them.

For a given value of an attribute, if the category is known (or if it matches a predefined regex), it returns the corresponding identifier. In case the category was never encountered before (frequent in the context of anomaly detection), it returns an integer corresponding to the category "Unknown". This corresponds to the function **TransformCategory** in Algorithm 1.

Normalizing Raw String Attributes. Strings, as found in security events, have their own syntax and semantic. As such, it is possible to apply Natural Language Processing (NLP) techniques to handle them [8]. The approach chosen for our auto-encoders requires that a sentence is represented as a raw array of integers, and this transformation is performed by the function **StringToIntegerArray** in Algorithm 1. In NLP, the interpretation is the following: every character of a string (the UTF-8 code of the character) is a word, and the string is a sentence.

3.3 Neural Network Structure

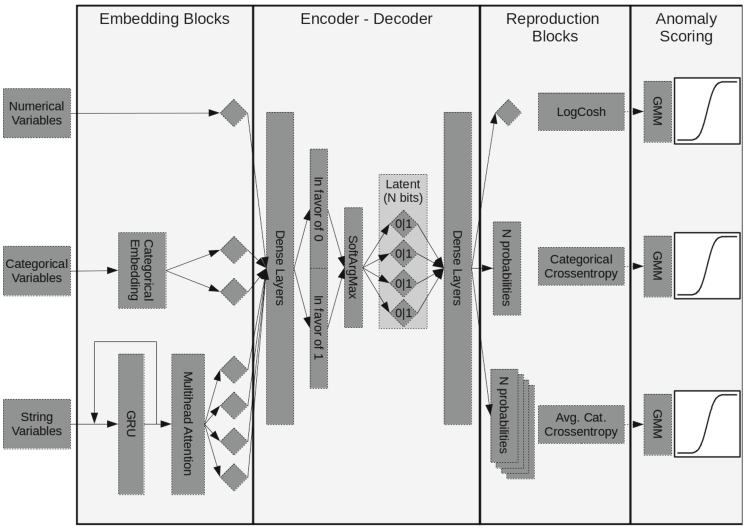

Fig. 2. Our proposed neural network structure

Given the diversity of inputs, each attribute needs to be processed differently by the auto-encoder. For the embedding part, the first layers are organized into independent blocks, one block for each input attribute. The output of each block is then horizontally concatenated before being processed by the shared layers of the encoder. Symmetrically, for the decoding part, the encoder outputs first pass through shared layers, and the last shared layer output is used as input to every reconstruction blocks.

We implemented three types of blocks (Embedding Blocks - Fig. 2), one for each type of variable (i.e., categorical, numerical or strings). For numerical values, the transformation corresponds to a one neuron layer. For categorical values, the input layer is composed of as much neurons as possible categories. The input block for strings is more complex. The structure chosen is inspired by state of the art NLP techniques: a combination of Gated Recurrent Units (GRU) neural networks [4] and multi-head self attention [25]. GRU are designed to learn long term dependencies between steps of a sequence. Attention mechanism highlights the most relevant steps inside a sequence. When combined together, they can compute pertinent low dimension representation of long sequences. However, these techniques have a high computational cost. Therefore, when the number of different strings is limited or when it is possible to find regex that reduces this number, strings should be handle as categorical variables.

3.4 Event Clustering

In the cybersecurity domain, regrouping similar events can ease the analysis process (i.e., analyzing a few groups of events instead of all events one by one). In addition to its interesting properties for anomaly detection, the auto-encoder can also provide a low dimension representation of its inputs thanks to its latent layer. This latent layer can be used for event clustering. Recent work [6] obtained good clustering performance by combining auto-encoders with Gaussian mixture model. In the case of security events, we found that the dominance of categorical variables leads to many clusters with a standard deviation close to 0, which is not an ideal case for Gaussian models (due to the division by the standard deviation in the Gaussian equation). By outputting vectors of bits (either 0 or 1), the approach we propose better fits the discrete nature of the variables found in security events (Fig. 3).

$$f(x_0, x_1) = \frac{e^{\beta x_1}}{e^{\beta x_0} + e^{\beta x_1}} \tag{1}$$

Fig. 3. Soft-ArgMax for binary variables. β controls the steepness

The latent layer is divided into 2 blocks of N components. For each of the component if the value of the first block is higher than the value of the second block, the corresponding component in the latent space will be close to 0. Otherwise, the component value in the latent space will be 1. We need to use the Soft-ArgMax function (Eq. 1) so that the output of the latent layer is either close to 0 or close to 1, while still being able to compute a gradient (required for

neural networks). The output of the latent layer is therefore a vector of N bits. These N bits define a cluster identifier, that is later used to regroup the events with equal identifier (Subsect. 5.2) (Fig. 4).

3.5 Anomaly Score Computation

$$f(x, \hat{x}) = logcosh(\hat{x} - x) \qquad (2) \qquad\qquad f(x, \hat{x}) = -\sum_{i}^{N} x_i * log(\hat{x}_i) \qquad (3)$$

Fig. 4. Loss functions: x is the input value and $\hat{x}$ the output one. N is the total number of categories and x_i the probability of being the i^{th} category

Per Attribute Score. Different types of variables also means different ways of computing the reproduction error. For numerical values, we use the logarithm of the hyperbolic cosine (Eq. 2). It behaves like $x^2/2$ for small value of x (i.e., fits more gradually when loss is close to 0) while behaving close to $x - ln(2)$ when x is large, which avoids giving more importance to extreme values. As output for categorical variables, the auto-encoder expresses a probability of being every possible category. As a measure of the error, we use the categorical cross entropy (Eq. 3). Similarly to categorical variables, for each character of a string, the auto-encoder outputs a probability distribution over possible characters. We use the average categorical crossentropy across every character as the error function.

Due to the diversity of attributes and types of attributes, the reproduction error for one attribute is not directly comparable with the reproduction error of another attribute. As an example, if error E_0 for attribute a_0 ranges from 0 to 1 and error E_1 for attribute a_1 ranges from 1 to 1.5, $E_0 < E_1$ is not indicative of anything. To be able to provide hindsight of what attribute might have caused the anomaly, we need to be able to compare anomaly score between attributes (Fig. 5).

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_{O}^{x} e^{-t^2} dt \qquad (4) \qquad\qquad CDF(x) = \frac{1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)}{2} \qquad (5)$$

Fig. 5. Cumulative Distribution Function at value x for as Gaussian distribution parametrized by (μ, σ). The error function is denoted erf

During our experiments on multiple datasets and types of security events, we have found that for most attributes, a Gaussian mixture model is an appropriate approximation of the true distribution of the reproduction error on normal data. The Expectation-Maximization algorithm [5] is used to find the parameters (mean μ, standard deviation σ and probability ϕ for each Gaussian). We take the parameters (μ_0, σ_0) of the Gaussian with the highest μ (i.e., the distribution of the least normal events). The anomaly scoring function for a single attribute is defined as the Cumulative Distribution Function (CDF) of the Gaussian distribution parametrized by (μ_0, σ_0), which provide a score between 0 and 1 to each attribute of an input vector. A score close to 0 implies that the value of

the attribute is considered normal, and a score close to 1 means that the value is likely anomalous.

Global Anomaly Score. At this point, we have a score between 0 and 1 for each attribute of the events. However, different types of event (e.g., network and system events) may have different number of attributes. As a simple way of providing a final anomaly score for an event, we use the metric (6), which essentially computes the frequency of anomalous attributes inside an event. (Fig. 6)

$$S = \frac{1}{n_e} \sum_{i=1}^{n_e} \begin{cases} 1, & \text{if } a_i > T_i \\ 0, & \text{if } a_i \leq T_i \end{cases} \tag{6}$$

Fig. 6. Anomaly score S for an event with n_e attributes with a score of $(a_1, a_2, ..., a_{n_e})$. $(T_1, T_2, ..., T_{n_e})$ is a threshold vector whose attribute can be configured individually

Finding Suitable Per Attribute Thresholds. In our approach, before computing the anomaly score for an event (Eq. 6), we need to determine a set of thresholds $(T_1, T_2, ..., T_{n_e})$ (with n_e the number of attributes) above which attributes will be considered abnormal. To this end, we link the cluster identifier and the threshold. For a given cluster, we compute the 99^{th} percentile of the score for each attribute on normal data and use it as the threshold. During inference, it is possible to encounter never-before-seen cluster identifier, and in this case the threshold for each attribute will be the global average score of the attribute on normal data (i.e., independent from cluster identifier).

4 Implementation

4.1 Heterogeneous Events

The monitored system produces events by observing different layers: system layer, network layer and application layer. The observation at system layer consists in recorded system calls. These system calls can vary from an operating system to another (e.g., Linux or Windows). The system call level observation consists in executed processes and write access to files. This information is logged using the tool *auditd* for Linux. On Windows machine, we use Sysmon to log the executed commands and created files. At network layer we produce network events by inspecting network flow from OSI layer 2 (link) to Layer 5 (application level protocols). At network level, we use the Zeek tool [21] (the new name of the Bro tool). For application events, HTTP requests in the Apache HTTP server and Squid HTTP proxy logs are collected.

Formally, a logged event is an array of attributes whose values can be either a string, an integer or a float. Strings can be either handled as raw strings (e.g., a command and its arguments) or as categorical variables (e.g., the executable path), and integer as either numerical values (e.g., number of bytes) or categorical values (e.g., port number). Therefore, instead of string, integer or float,

we will consider that the type of an attribute of an event can be either categorical, numerical or string. Occasionally, attributes of an event might be missing (e.g., transport error, logging failure, etc.), and in that case, the event will not be analyzed. We will denote a type of event as a set of retained attribute and their corresponding type. For example the zeek DNS log event type can be defined as $\{source.ip : categorical, destination.ip : categorical, dns.query : string, dns.answers : string\}$. We need to create one auto-encoder model per event type. The complete list of event types is given in Subsect. 4.3.

4.2 The Monitored System Architecture

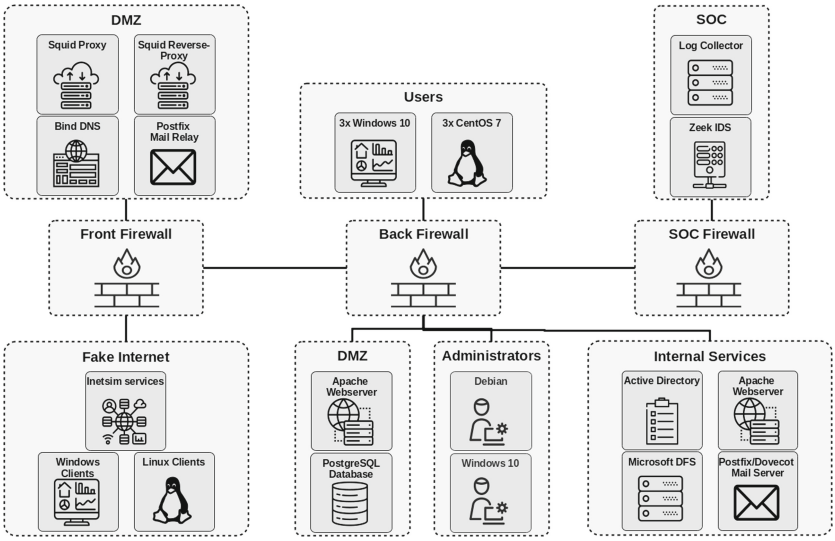

Fig. 7. Overview of the monitored system

For assessing our approach, we used virtual machines to deploy an Information System architecture reproducing the behavior of a small to medium sized company (Fig. 7). The monitored part of the system is composed of 20 machines that are distributed across 6 different VLAN separated by 3 firewalls. The Zeek IDS tool analyzes the Ethernet traffic of each of these VLAN and logs the connection and the DNS requests. Linux machines are based on CentOS 7. The Microsoft machines run on Windows Server 2016 for the servers and Windows 10 for the workstations. To simulate the activity of internal users connecting to the Internet, we use the tool InetSim to mimic a few external services (e.g., web, DNS). Multiple clients are also deployed outside the network and are browsing the company's public website. To generate the attack behaviour, we have intentionally introduced vulnerabilities inside the public website to facilitate exploitation.

4.3 Considered Event Types

There are four different sources of events that are used to monitor the security from the OS (Auditd, Sysmon), network (Zeek) and application (HTTP server and proxy logs) point of view. From each of these point of view, different aspects are considered (e.g., process and file monitoring by the OS) and require different sets of attributes, which we call event types. We organize the collected events into seven different event types and we describe them in the following list.

Auditd (Linux)

- *Executed process*: {command line: str., process working directory: cat., executable: cat., hostname: cat., effective id: cat., effective group id: cat., user id: cat., user group id: cat.}
- *Write access to files*: {hostname: cat., process working directory: cat., executable: cat., outcome: cat., syscall: cat., effective id: cat., effective group id: cat., user id: cat., group id: cat., file path: cat.}

Sysmon (Windows)

- *Executed commands*: {process args: str., process working directory: cat., executable: cat., parent executable: cat., hostname: cat., user id: cat., group id: cat.}
- *File creation*: {hostname: cat., process working directory: cat., executable: cat., file path: cat., user id: cat., group id: cat.}

Zeek

- *DNS*: {source IP: cat., destination IP: cat., dns query: str., dns answers: str.}
- *Network connection*: {source IP: cat., destination IP: cat., destination port: cat., network transport: cat., duration: num., total response bytes: num., total origin bytes: num.}

Apache and Squid Logs

- *HTTP requests*: {hostname: cat., method: cat., status_code: cat., source address: cat., url: str., user agent: str.}

5 Approach Assessment

5.1 Collecting Data

To perform the learning phase of the auto-encoders, we must collect logs corresponding to the normal activity of the monitored system. The recorded activity is composed of users of the company regularly browsing both the internal and the public website of the company (available from the external network). They also exchange emails inside and outside the company. Users create, edit and delete documents and share some of these documents with other users through the company's shared directories. Additionally, administrators regularly perform actions (e.g., configuration changes). Finally, multiple clients from the simulated Internet zone browse the public server. User and client actions have been automated

using tools that directly manipulate the user interface so that they effectively generate events that can be seen in system logs. The administrative tasks have been performed manually.

Training Dataset. The generated dataset is composed of 8 738 080 normal events and correspond to a month of activity. 1 017 682 (11.6%) are generated by auditd, 29 992 (0.3%) by Sysmon, 7 013 337 (80.3%) by Zeek and 677 069 (7.7%) are HTTP requets. As these different types of events have different attributes, a model has to be learnt for each of these types, thus, we separate the dataset in 7 sub-datasets. Each of these sub-datasets is randomly divided in 3 parts. 60% of the data is used for training the model. 20% to periodically evaluate the performances of the model on unknown data and stop the training once the performances degrade (early stopping), which help combat over-fitting and reduce the number of false positives. The remaining 20% are used to adapt the different parameters (Sect. 3.5) and finalize the model on never-before-seen data. This separation in three sub-datasets is common for neural network training. We train one model for each of the 7 event types.

Attack dataset. During a business day, the following attack is performed:

1. The attacker crawls the public web server of the company in order to find a potential vulnerability;
2. He exploits a vulnerability on the public web server of the company;
3. Then, he can modify a page restricted to the moderators of the web site;
4. A moderator visits the page from inside the company network, CVE-2018-8495 is exploited and the user unwisely accepts the execution of the script;
5. The script deploys a remote access trojan on the machine and contacts the command and control server;
6. The attacker stealthily scans a few hosts and finds the file server;
7. A file is downloaded from the file server and uploaded to the attacker's server;
8. The attacker erases its tracks and leaves the company's network.

The resulting dataset contains 298 302 events with both normal and abnormal activities. In total, around 1500 events are related to the attack (0.5%).

5.2 Assessing Clustering Capabilities

The proposed auto-encoder provides a cluster identifier for each event. We can use this identifier to regroup events together to lower the total number of alerts an analyst has to investigate. On the attack dataset, we obtain a total of 191 different cluster identifiers. By design, the auto-encoder tends to project unknown data points closer to normal points in the latent space. Therefore, a normal event and an anomaly can share the same cluster identifier. To avoid mixing normal and anomalous events inside a group, events are regrouped if they have the same cluster identifier as well as the same anomaly score (Eq. 6). With these conditions, we regroup the 298 302 events of our attack dataset into 293 groups.

In a typical IT system, a significant change to the system (e.g., new user) can lead to repeated and similar false positives. Inside our attack dataset, we identified approximately 10500 of these false positives and they are regrouped into 84 different clusters (125 events per cluster on average). For our attack scenario, we identified 1500 events related to the attack, and they are regrouped into 28 different clusters (54 events per cluster on average). This confirms that the chosen clustering approach helps reduce the number of false positives to be analyzed without diluting relevant attack information inside large clusters.

5.3 Anomaly Detection Results

The size reduction, detailed in the previous section, allowed us to manually annotate the dataset and we found that the original 0.5% of anomaly in the attack dataset are now distributed in 28 groups (9.56%). From there we can compute metrics relative to the performance of our approach (Fig. 8).

$$P = \frac{TP}{FP + TP} \quad (7) \qquad R = \frac{TP}{FN + TP} \quad (8) \qquad F_1 = 2 \times \frac{P \times R}{P + R} \quad (9)$$

Fig. 8. Precision (P), Recall (R) and F_1 score

Due to the imbalance in class distribution (only 9.56% of positives), we chose to use precision (7) that correspond to the ratio of true positives among all the events classified as anomalies, the recall (8), also known as the true positive rate, and F_1 score (9) metrics, which is the harmonic mean of the recall and the precision. For all these metrics, a value of 1 implies a perfect classifier, and a value of 0 a useless one. In a context of a prioritisation tool, analysts will define a maximum number of alerts that they can handle in a day. For this reason, we provide the recall (i.e., proportion of anomalies accurately identified) when considering the top 100, 50 and 10 groups (with regard to their anomaly score). We provide the results in Table 1. We also found a false positive rate of 0.12% on normal data with the threshold set as the minimum score of attack related events (i.e., 100% true positive rate).

Table 1. F_1 score, precision and recall for the top 10, 50 and 100

Top N groups	F_1	Precision	Recall
10	0.162	0.333	0.107
50	**0.5**	0.342	0.929
100	0.44	0.28	**1**

More specifically, from a threat hunter perspective, analyzing the top 10 groups of events would be enough to determine that arbitrary code have been executed on the web server and on the infected Windows machine. Analyzing the top 50 events is enough to reconstruct the major steps of the attack and identify the attacker IP address as an Indicator of Compromise (IoC).

6 Conclusion

In this paper, we proposed a method that relies on neural networks auto-encoders to compute anomaly scores for heterogeneous events prioritisation. We propose an original approach to exploit the latent space of the auto-encoders to provide clustering capabilities. This is used to reduce the volume of information that is presented to the analysts by regrouping similar events together.

We drew inspiration from state-of-the-art deep learning techniques to handle the most common attribute types found inside security events (numerical, categorical and string attributes). These techniques simplify the design of the feature extraction and transformation processes that are required by any machine learning-based approach. This allowed us to quickly create a specific model for each event types inside our dataset. While other machine learning methods for anomaly detection have already been proposed, they need specific feature engineering for each new type of events, which requires knowledge in data science that is rarely available among Security Operation Center (SOC) analysts.

Using our method, we regrouped the 298 302 events of our test dataset in 293 groups. When manually analyzing these groups, we found all the attack-related events within the 100 groups with the highest anomaly scores.

The ability to prioritize heterogeneous events is the first step towards behavioural anomaly-based attack detection tools for SOC. Future work will focus on anomaly contextualisation using automated correlation techniques, as well as visualization techniques to further simplify the investigation process for analysts.

References

1. Akoglu, L., Tong, H., Vreeken, J., Faloutsos, C.: Fast and reliable anomaly detection in categorical data. In: Proceedings of the 21st ACM international conference on Information and Knowledge Management, pp. 415–424. ACM (2012)
2. Bellman, R.E., Dreyfus, S.E.: Applied Dynamic Programming. Princeton University Press, Princeton (2015)
3. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 93–104 (2000)
4. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
5. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc.: Ser. B (Methodol.) **39**(1), 1–22 (1977)
6. Dilokthanakul, N., et al.: Deep unsupervised clustering with Gaussian mixture variational autoencoders. arXiv preprint arXiv:1611.02648 (2016)
7. Ding, Z., Fei, M.: An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proc. Volumes **46**(20), 12–17 (2013)

8. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1285–1298. ACM (2017)
9. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd **96**, 226–231 (1996)
10. Hawkins, S., He, H., Williams, G., Baxter, R.: Outlier detection using replicator neural networks. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2002. LNCS, vol. 2454, pp. 170–180. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46145-0_17
11. He, Z., Xu, X., Huang, J.Z., Deng, S.: A frequent pattern discovery method for outlier detection. In: Li, Q., Wang, G., Feng, L. (eds.) WAIM 2004. LNCS, vol. 3129, pp. 726–732. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27772-9_80
12. Kaastra, I., Boyd, M.: Designing a neural network for forecasting financial and economic time series. Neurocomputing **10**(3), 215–236 (1996)
13. Kriegel, H.P., S hubert, M., Zimek, A.: Angle-based outlier detection in high-dimensional data. In: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 08, p. 444. ACM Press (2008). https://doi.org/10.1145/1401890.1401946
14. Lanoe, D., Hurfin, M., Totel, E.: A scalable and efficient correlation engine to detect multi-step attacks in distributed systems. In: 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), pp. 31–40, October 2018. https://doi.org/10.1109/SRDS.2018.00014
15. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422. IEEE (2008)
16. Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., Meng, D.: Log2vec: a heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1777–1794 (2019)
17. Mikolov, T., Chen, K., Corrado, G.S., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013)
18. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: an ensemble of autoencoders for online network intrusion detection. arXiv preprint arXiv:1802.09089 (2018)
19. Pang, G., Ting, K.M., Albrecht, D.: Lesinn: detecting anomalies by identifying least similar nearest neighbours. In: 2015 IEEE International Conference on Data Mining Workshop (ICDMW), pp. 623–630. IEEE, November 2015. https://doi.org/10.1109/ICDMW.2015.62
20. Pascoal, C., De Oliveira, M.R., Valadas, R., Filzmoser, P., Salvador, P., Pacheco, A.: Robust feature selection and robust pca for internet traffic anomaly detection. In: 2012 Proceedings IEEE Infocom, pp. 1755–1763. IEEE (2012)
21. Paxson, V.: Bro: a system for detecting network intruders in real-time. Comput. Netw. **31**(23–24), 2435–2463 (1999)
22. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Elsevier, Amsterdam (2014)
23. Schölkopf, B., Williamson, R.C., Smola, A.J., Shawe-Taylor, J., Platt, J.C.: Support vector method for novelty detection. In: Advances in Neural Information Processing Systems, pp. 582–588 (2000)

24. Shen, Y., Stringhini, G.: Attack2vec: leveraging temporal word embeddings to understand the evolution of cyberattacks. In: 28th {USENIX} Security Symposium Security 2019, pp. 905–921 (2019)
25. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
26. Veeramachaneni, K., Arnaldo, I., Korrapati, V., Bassias, C., Li, K.: Aî 2: training a big data machine to defend. In: 2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), pp. 49–54. IEEE (2016)
27. Wong, W.K., Moore, A.W., Cooper, G.F., Wagner, M.M.: Bayesian network anomaly pattern detection for disease outbreaks. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03), pp. 808–815 (2003)
28. Zhou, C., Paffenroth, R.C.: Anomaly detection with robust deep autoencoders. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 665–674. ACM (2017)