OutGene: Detecting Undefined Network Attacks with Time Stretching and Genetic Zooms*

Luís Dias[1,2][0000−0003−2842−6655], Hélder Reia[1,2], Rui Neves[3][0000−0001−5482−9883], and Miguel Correia[2][0000−0001−7873−5531]

[1] CINAMIL, Academia Militar, Instituto Universitário Militar, Portugal
{dias.lfxcm,reia.hf}@mail.exercito.pt
[2] INESC-ID, Instituto Superior Técnico, Lisboa, Portugal
[3] Instituto de Telecomunicações, Instituto Superior Técnico, Lisboa, Portugal
{rui.neves,miguel.p.correia}@tecnico.ulisboa.pt

Abstract. The paper presents OutGene, an approach for streaming detection of malicious activity without previous knowledge about attacks or training data. OutGene uses clustering to aggregate hosts with similar behavior. To assist human analysts on pinpointing malicious clusters, we introduce the notion of *genetic zoom*, that consists in using a genetic algorithm to identify the features that are more relevant to characterize a cluster. Adversaries are often able to circumvent attack detection based on machine learning by executing attacks at a low pace, below the thresholds used. To detect such stealth attacks, we introduce the notion of *time stretching*. The idea is to analyze the stream of events in different time-windows, so that we can identify attacks independently of the pace they are performed. We evaluated OutGene experimentally with a recent publicly available dataset and with a dataset obtained at a large military infrastructure. Both genetic zoom and time stretching have been found to be useful, and high values of recall and accuracy were obtained.

Keywords: Time Stretching · Genetic Zooms · Intrusion Detection · Security Analytics

1 Introduction

The exponential growth of data and of its value makes data assets mission-critical to many organizations [38]. The increasing occurrence of cybercrime and, generically, of cyberattacks [8], raises the need for better methods of protecting computers and the information they store, process and transmit. This objective is challenging as, for example, the average time a company takes to detect certain attacks, Advanced Persistent Threats (APTs), is about 100 days [32].

Intrusion Detection Systems (IDS) have been proposed as an attempt to deal with this increasing number of attacks, that often manage to elude existing

* This research was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UID/CEC/50021/2019 (INESC-ID), by the Portuguese Army (CINAMIL), and by the European Commission under grant agreement number 830892 (SPARTA). We warmly thank prof. Victor Lobo for feedback on a previous version of this work.

protections [11, 12]. Most IDSs are either *signature-based* (search for known attack patterns) or *anomaly-based* (detect deviations from baseline behavior), but both approaches have limitations: knowledge about attack patterns tends to be incomplete, as new attacks and attack variants are constantly appearing [43]; and anomaly-based detection requires clean training data, i.e., data of normal operation without attacks, to train the IDS, which is hard to obtain in systems in production. Moreover, anomaly-based IDSs have to discover attacks hidden among what may be a huge amount of data representing normal behavior [58]. IDSs can also be classified as network-based (that inspect communication data) or host-based (that inspect host activity). Finally, IDSs can be classified as online (detect intrusions in runtime) and offline (detect intrusions later, when decided by someone). This paper is about *online network-based intrusion detection*.

Machine learning (ML) approaches for intrusion detection have been receiving much attention [4,5,13,20,29,58]. Handling high volumes of security-relevant data is unfeasible for humans, so ML techniques can come to assistance. An example approach are the above-mentioned anomaly-based IDSs, with their drawbacks [4]. A more recent ML approach to intrusion detection uses *clustering and/or outlier detection* to identify entities – typically users or hosts – that have an anomalous behavior [7, 18, 40, 55, 56]. To be precise, the approach does not detect intrusions, but anomalies that have to be further diagnosed as intrusions or some other sort of anomaly. This approach is interesting because it does not require knowledge about attacks (signatures/rules) or clean training data. However, it brings in two difficulties. First, a human analyst has to inspect the outliers or suspect clusters, which is a non-trivial task, although it is also necessary in anomaly-based detection and even signature-based detection (although in this latter case the attack is already labelled with a class). Second, attackers can often circumvent ML-based attack detection by executing attacks – e.g., port scanning – at a low pace, below the thresholds used.

We present OutGene, a network intrusion detection approach that detects attacks that are undefined (no signatures) without clean training data. This contrasts to both signature-based detection, that needs attack signatures, and anomaly-based detection, that needs clean training data. OutGene does clustering of hosts with similar behavior and detects ouliers, leveraging the assumption that hosts that are doing attacks behave in a way that is distinguishable from the others. OutGene does online, *streaming*, attack detection, i.e., it does detection continuously, not by processing bulk data sporadically.

OutGene solves the two difficulties mentioned above. First, to assist human analysts on pinpointing malicious clusters, we introduce the notion of *genetic zoom*. The idea is to use a genetic algorithm to identify the best subset of features that provides the same clustering output as the full set of features. For example, genetic zoom might say that out of 26 features, only a certain subset of 8 features is relevant to obtain the clusters. Examples of such features might be the number of ports used by an entity and the number of ports contacted by an entity. It is much easier to understand what was the malicious behavior by inspecting the values of 8 features instead of 26.

Second, to detect *stealth attacks* that try to pass below the radar of the detection scheme, we introduce the notion of *time stretching*. The idea is to analyse the stream of events in different time-windows, at different time scales, so that we can detect attacks independently of the pace at which they are executed (e.g., a slow network scan). For example, an attack may be detected if we analyse traffic at the scale of one hour, but not at the scale of one day or one minute.

We implemented the proposed approach as a system that we also designate OutGene. This system is based on a set of large-scale data processing and storage packages. Incoming data, e.g., network flow data, is consumed by Apache Kafka [28], that stores it using the Hadoop Distributed File System (HDFS) [42]. The first is used to decouple processing from data producers and the former as long-term storage and checkpoint location (for the aggregation process). Apache Spark [34] consumes data incoming from Kafka and does most of the analysis.

Much research in machine learning-based intrusion detection uses synthetic datasets [4]. Instead, we aim that OutGene works in real settings so we evaluated it using real network data from two real-world networks: one dataset that is publicly available [50] and another from an administrative network of a large military infrastructure that we collected for this work. Moreover, we emulated stealth attacks, which may pass unnoticed by traditional detection systems, to evaluate our approach on such attacks. The obtained results in both datasets reveal that there are significant improvements by analyzing different time-windows as well as by having outlier explanation provided by *genetic zoom*. The use of real network data was challenging due to data noise, lack of full context, and lack of labels. Nevertheless, high values of recall and accuracy were obtained with the military network dataset (near 1).

The main contributions are: (1) a practical approach for online detection of network attacks that uses clustering and outlier detection to avoid the need of knowledge about attacks and clean training data, implemented and tested with real-data datasets; (2) the genetic zoom mechanism, which uses a genetic algorithm to help the human analyst; (3) the time stretching mechanism, that analyses traffic in different time frames, in order to detect stealth attacks.

2 OutGene Approach Overview

OutGene does not rely on knowledge about what is bad behavior as in signature-based methods, or what is good behavior, as in typical anomaly detection. Hence, our approach does not use supervised learning that needs training data, nor rules or thresholds, which can be easily circumvented by attackers or are sensitive to new systems deployed in a network.

Inspired by [7, 18, 56], OutGene uses unsupervised learning, more specifically, a clustering algorithm to group entities with similar behaviors. The term *entity* designates something that is identified by an IP address: typically a host but there are other possibilities, like a network behind NAT [45]. Furthermore, OutGene uses feature selection jointly with clustering to obtain a better description (i.e., the most relevant features) of anomalous groups (i.e., of small

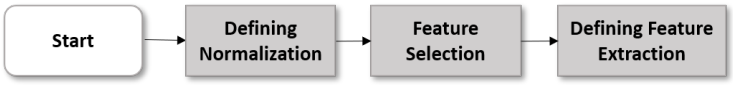

Fig. 1. Flowchart of the pre-runtime phase

clusters). The main goal of the developed work is to extract useful information to detect and characterize cyberattacks from *streaming* data, without information of previous patterns. Since most of the used features are count-based (e.g., count of bytes sent), the approach does time-based aggregation of the events in real time through incremental aggregation [24].

OutGene aims to *detect attacks* and *select the relevant features* to: classify them by performing clustering to extract information from network flows (that we will designate *netflow* as it is typically Netflow data [10], although it may also be extracted from raw packet data) using *generic features* that are considered relevant in the scope of security; provide improved insight on outliers using *genetic zoom*, a genetic algorithm to search for the *specific features* that characterizes outliers (i.e., small clusters) in our set of data; do detection using *time stretching*, i.e., repeating the previous steps on different time-windows. By doing this, it is possible to get more knowledge about the data and apply this knowledge to classify and label specific behaviors.

The approach is divided in two phases. The *pre-runtime phase* consists in exploratory data analysis to prepare the *runtime phase*, when OutGene continuously processes data to detect attacks. They are explained in detail next.

Pre-runtime phase The pre-runtime phase is executed before the system is deployed to define generic feature extraction and normalization (see Figure 1). This phase consists in exploratory data analysis techniques and can be detailed in three steps: the definition of the normalization of the data, to transform the data so that we know what parameters to normalize to make all data consistent; the feature selection, to know which features are going to be extracted to characterize the available data, as well as the time periods (to aggregate time-windows) to extract the features; the definition of how the features are extracted, given the kind of data considered.

Runtime phase The runtime phase corresponds to detection in normal operation (see Figure 2). The stream of data that is processed contains the traffic flows that are collected (e.g., by routers) and passed to OutGene. The processing starts by generic feature extraction and normalization. The extracted features are given as input to the clustering algorithm which groups entities (typically hosts) with similar behavior.

If clustering produces outliers, there are two options. If this kind of outlier/anomaly has been observed before, which is the typical case in cruise speed, it can simply be reported. Otherwise, manual intervention by a human analyst is required, which is inevitable when the possibility of unknown attacks is considered.

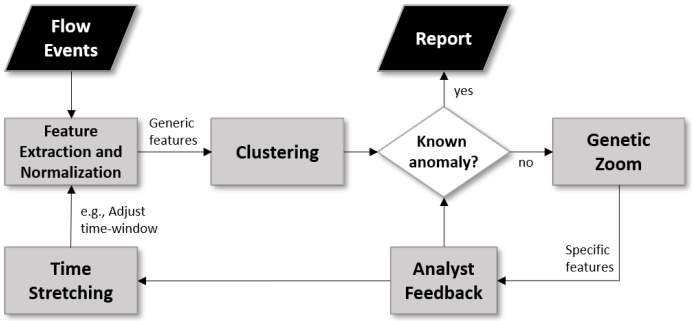

Fig. 2. Flowchart of the runtime phase (notice the *genetic zoom* and *time stretching* processes, respectively on the right and bottom-left)

The purpose of *genetic zoom* is to assist the human analyst on diagnosing malicious clusters (among the outliers only). The idea is to use a *genetic algorithm* [17] to identify the features that are more relevant to characterize a cluster. The genetic algorithm is used to understand which features are important to identify an entity as belonging to that cluster. Moreover, it helps understanding which features better characterize the data in general, discarding features that are less important, i.e., that do not change the clusters that are obtained, at a certain iteration of the loop. The pre-runtime phase involves defining features that are potentially useful to pinpoint outliers. However, not all of these features may be relevant at each iteration of the loop in runtime. Hence, choosing a subset of the original features with the genetic algorithm will lead to better knowledge of the anomaly.

The second mechanism we introduce is *time stretching*. Figure 2 shows a single loop in which features are passed to clustering, then clusters are diagnosed automatically or, in some cases, manually. However, in reality there are several loops executed in parallel processing flows corresponding to several time-windows (e.g., 10 min., 30 min., 1 hour, 4 hours, 8 hours, 1 day), leveraging incremental aggregation. Hence, OutGene is not limited to bulk processing of large data corresponding to long periods. This processing in different time-windows is what we designate time stretching and what allows detecting both conspicuous attacks quickly (e.g., DDoS) and stealth attacks (e.g., slow port scan).

3 The OutGene Platform

This section presents the basic OutGene platform. The details about clustering, genetic zoom, and time stretching are deferred for Section 4. To handle the processing of large volume of flows, we propose a platform based on distributed stream processing and analytics modules. The stream processing module has the objective of handling the data that arrives from data sources, as well as enabling checkpointing for streaming aggregation allowing time stretching. The analytics

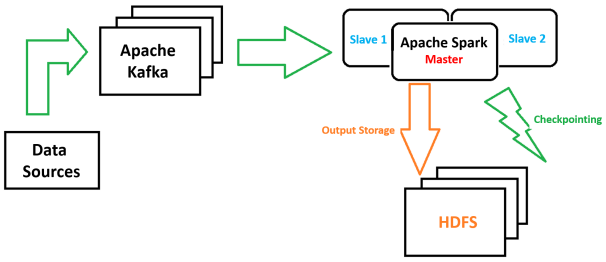

Fig. 3. System architecture

module has the objective of extracting the features and to execute the clustering and the genetic zoom algorithms. The architecture is represented in Figure 3).

Stream processing module The stream processing module decouples data streams from the analytics module. To do so, the Apache Kafka [28] framework and HDFS [42] were used. Apache Kafka is a distributed streaming platform, with capabilities to publish and subscribe streams of records, similar to a message queue or enterprise messaging system; To accomplish this, Kafka stores streams of records in categories called topics in a fault-tolerant way. A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber, that is, a topic can have zero, one, or more consumers that subscribe to the data written to it. Kafka topics are divided in partitions. Partitions allow us to parallelize a topic by splitting the data in that particular topic across the multiple servers. In order to guarantee the correct operation of the brokers (servers), a broker manager is needed. This service can be implemented with the Apache Zookeeper coordination service [23]. For the experiments we created a Kafka topic with replication factor 3 (i.e., all data is replicated in 3 nodes), where the data streams corresponding to netflow data were inserted. Regarding HDFS, its main role is to store checkpoints (for the aggregation process) as explained later, although it might also be used for long-term storage.

Analytics module The analytics module is where all the computation is made. To implement it, the Apache Spark framework was used as a Kafka consumer. Apache Spark is a distributed and highly scalable in-memory data analytics system with four main submodules: Spark SQL, Spark Streaming, MLlib and GraphX. For OutGene, only the Spark Streaming and MLlib modules are necessary. The Spark MLlib module is able to run advanced data analysis algorithms in a scalable way. In its core, Apache Spark provides the runtime for in-memory massive parallel data processing, and different parallel machine learning libraries are running on top of it. Apache Spark distributes tasks over multiple computer nodes (i.e., worker nodes), although even on a single node it can spill data to disk avoiding the main memory bottleneck [27]. Regarding Spark Streaming, it is able to process continuous streams of data in real time, with the functionality of checkpointing and windowing. Later we explain how it is used. Every Spark ap-

plication consists on a driver program that runs the main function and executes various parallel operations in a cluster. The main abstraction Spark provides is the *resilient distributed dataset* (RDD), which is a collection of elements — similar to an array — partitioned across the nodes of the cluster that can be operated on in parallel. It is possible to ask Spark to persist RDDs in memory, allowing them to be reused efficiently across parallel operations.

4 OutGene Approach Instantiation

The OutGene approach can be instantiated in different ways. An instantiation entails four aspects: feature extraction, clustering, feature selection with a genetic algorithm (i.e., genetic zoom), and online mode processing of different time-windows (i.e., time stretching). We present each of the aspects next.

4.1 Feature extraction

Selecting the features to use is a crucial step, because they provide the symptoms that allow distinguishing normal traffic from attacks. The Spark transformations and operations needed to perform feature extraction can be divided in three steps: (1) a *Map* transformation, to group pairs of values; (2) the *CountByValue* operation for features that count the frequency of an occurrence (e.g., number of connections made); (3) the *reduceByKey* operation, to remove the repeated values by summing the values for each entity (e.g., a computer, IP, user).

The selected features were extracted from network flows. The choice of such data to implement OutGene was based on good results in related work [21, 40, 44]. A set of 26 features was used, split in two groups: the first half are features about the source computer (i.e., the source IP address) and the other half are about the destination computer (destination IP address). The first half features are shown in Table 1. The other 13 features are similar (e.g., DConn connections received, and DPSum packets received). In the table, it is clear that only four application-layer protocols are considered explicitly in the features: HTTP, IRC, SMTP, and SSH. We selected these protocols based on the literature, instead of selecting a larger range (not trivial as there are around 1000 well-known ports plus 10000 reserved ports) or selecting the ports used in the datasets (which would be a form of bias). We decided to stick to what the literature says is meaningful and observe the results, which were quite positive (Section 5.3). Features are extracted for a given time-window (e.g., 10 min, 1 hour, 1 day).

4.2 Clustering algorithm

Our approach aims to differentiate well-behaved from misbehaving entities, so the clustering algorithm has to separate entities with different behavior, being different behavior expressed by different values of features. From the clustering algorithms available in Apache Spark MLlib, we have chosen *K-means* [31] for two reasons: (1) it splits data points into a predefined number of clusters $\mathcal{K}$,

Table 1. Features extracted for a source IP

Feature	Description
SConn	Number of connections made
SPus	Number of ports used in source
SPcon	Number of ports contacted by a source
SPsum	Sum of packets sent by a source
SP80t	Sum of packets sent by a source to port 80 (HTTP)
SP80f	Sum of packets sent by a source from port 80 (HTTP)
SP194t	Sum of packets sent by a source to port 194 (IRC)
SP194f	Sum of packets sent by a source from port 194 (IRC)
SP25t	Sum of packets sent by a source to port 25 (SMTP)
SP25f	Sum of packets sent by a source from port 25 (SMTP)
SP22t	Sum of packets sent by a source to port 22 (SSH)
SP22f	Sum of packets sent by a source from port 22 (SSH)
SBytes	Sum of bytes sent by a source

which is important to force the appearance of small clusters with outliers; (2) K-means is known to be a good option when the number of samples is large (e.g., more than 10,000), which is our case.

K-means works iteratively, assigning each data point to one of $\mathcal{K}$ groups based on the distance to the group's centroid. The distance metric used is often the Euclidean distance, which is also the one we used in practice. At the end, data points with similar features are in the same cluster. The output of this algorithm is the number of the cluster to which each entity belongs to, and the central point (centroid) of each cluster.

The features have to be normalized using min-max normalization before clustering is performed. Interestingly we also did experiments with logarithmic normalization, but the detection results were much worse, as this form of normalization mitigates the differences between outliers and normal behavior.

4.3 Genetic zoom

The genetic zoom mechanism is based on a *genetic algorithm* [17]. The goal is to find the best subset of features that provides the same clustering output as generic features (i.e., initial clustering). The genetic zoom scheme is based on a wrapper model [3] that iteratively (1) selects subsets of features and (2) evaluates clustering quality using the selected subset (see Figure 4). Given that an exhaustive search of the $2^{\mathcal{D}}$ possible feature subsets (where $\mathcal{D}$ is the number of generic features) is intractable for high dimensionality, we choose as search strategy a genetic algorithm to provide a near-optimal response, by selecting a acceptable subset of features (i.e., the fitness function output stabilizes). Notice that the analyzed clusters are only the outliers, that are more likely to be the entities with misbehavior. The analyst either reports known anomalies or searches for new ones.

Genetic algorithms entail four concepts: gene, individual, population, and generation. A gene is a property that characterizes an individual. An individual is a candidate solution to the problem that one wants to solve with the algorithm. An individual is characterized by a set of genes. A population is a set

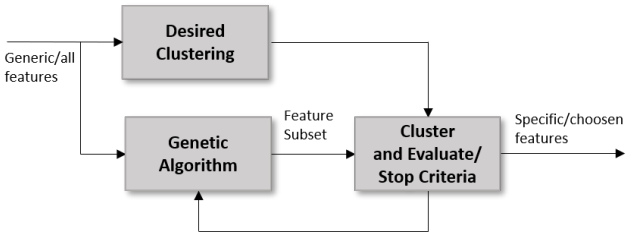

Fig. 4. The wrapper model for feature selection

of individuals, which can be modified from generation to generation. A generation is a new population, inheriting individuals from previous populations or modifying them (e.g., with operators such as crossover and mutation).

In this work, 26 genes were used, corresponding to the 26 features. Since we want to reduce the number of features, each gene can have the value 1 or 0 corresponding to a feature being active or not for that individual. The *zoom* consists in reducing the number of active features/genes (number of 1s in the individual's array of features) to a number $\mathcal{F}$ lower than 26. $\mathcal{F}$ will be the lowest possible as long as there is an individual (i.e., set of features) that reproduces the same outliers as the clustering with all the features.

The selection of individuals used to breed a new generation is made using a fitness function. We use maximizing fitness function to get higher value possible (details below). The fitness function does the following:

1. Reduce the number of genes/features that are active (i.e., set to '1') to $\mathcal{F}$;
2. Re-execute the clustering algorithm with the selected features;
3. Only if the same outliers (i.e., clusters with one entity) are present, calculate the similarity between new output and initial clustering using the rand index adjusted for chance [22];
4. The similarity score is returned (a value between 0 and 1).

After getting this evaluation, the best individuals (the ones that obtained highest similarity score) are selected for the next generation. To create a new population, two operators are used: crossover and mutation. The crossover operator requires two individuals that came from the previous generation, called the parents, and the offspring is created by exchanging genes of parents among themselves until the crossover point is reached. After the crossover operator is executed, the mutation operator is applied. Mutation is applied to a new offspring formed to change their genes given a low random probability. It occurs to maintain diversity within the population and prevent premature convergence. Both the crossover and mutation operator use functions that shuffle the attributes of the input individual and return a mutant.

In this work individuals have a set of 26 genes (i.e., total number of features). A population has $\mathcal{N}$ individuals, being the first generation generated randomly. There is a total of $\mathcal{G}$ generations and each generation is created with half of the best individuals from the previous generation plus a set of individuals generated

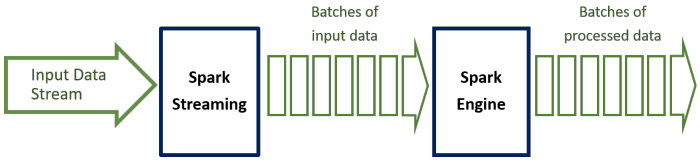

Fig. 5. Spark streaming process

with crossover and mutation operations. This way, we keep our best individuals and mutate from them to try to obtain better individuals. In the experiments we have used $\mathcal{N} = 32$ and $\mathcal{G} = 40$ which revealed to achieve good results efficiently.

The genetic algorithm was implemented using an evolutionary algorithm framework – the Distributed Evolutionary Algorithms in Python (DEAP) [16] –, as OutGene was mostly implemented in Python. DEAP is a recent evolutionary computation framework for rapid prototyping and testing of ideas, seeking to make algorithms explicit and data structures transparent. The genetic algorithm can be fully configured through DEAP by configuring the fitness function and the existing operators. The configurations made were based on the GENITOR scheme [52]: when selecting the individuals for a new generation, the best individuals of the previous generation are kept.

4.4 Time stretching

OutGene analyses data on different time-windows, what we call time stretching, to allow detecting stealth attacks. For that purpose we leverage the concept of *incremental aggregation*, which consists in executing aggregate functions (e.g., count or sum) continuously over streams of data [24]. The aggregate functions are used to calculate the values of the features.

To avoid the effort of executing the aggregate functions several times in parallel, we execute them on a *base time-window* of duration $\mathcal{B}$. In practice we considered $\mathcal{B} = 10min.$. Then, for the larger time-windows, OutGene simply executes the aggregate functions over the results obtained for $\mathcal{B}$. For example, whenever base time-window number i finishes, the system calculates the features not only for $\mathcal{B}_i$ of 10 min., but also for the time-window of 30 min. using the results of the last 3 last base time-windows $(\mathcal{B}_{i-2}, \mathcal{B}_{i-1}, \mathcal{B}_i)$. The same idea is applied for the larger time-windows.

Time stretching has been instantiated leveraging the incremental aggregation supported by the Spark Streaming module. This module works as shown in Figure 5, receiving live input data streams and dividing the data into mini batches. The mini batches are then processed, by the clustering algorithm in OutGene.

The operations supported by Spark Streaming are similar to those of Apache Spark, but with a time parameter that has to be defined. For feature extraction, the main operation used was *reduceByKeyandWindow*, that is similar to *reduceByKey* (used in batch mode) but with two more parameters: window length, the total duration of the window, and sliding interval, the interval at which the

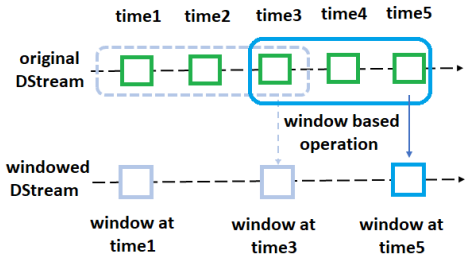

Fig. 6. Spark streaming window operations (adapted from [1])

Table 2. Summary of the dataset characteristics

Dataset	Size	Num. events	Num. hosts
LANL	1.1 GB (compressed)	129,977,412	12,027
Military	160 GB		5,500

window operation is performed. As shown in Figure 6, every time the window slides over a source data stream (DStream), the source RDDs that fall within the window are combined and operated upon to produce the RDDs of the windowed DStream.

For the experiments we set up an HDFS directory to store Apache Spark streaming information (checkpointing) necessary for window aggregation. Apart from the checkpointing data, the outputted data (extracted features and clustering results) was also saved on a HDFS directory to keep the progress registered.

5 Experimental Evaluation

This section presents the experimental evaluation of the OutGene instance we created, focused on the genetic zoom and time stretching mechanisms. Despite the existence of related approaches in the literature (see Section 6), implementations of those closer to ours are not available, so we present no experimental comparisons.

We first present the datasets, then the evaluation itself. As mentioned before, we used three servers, designated simply host 1 (the master), host 2 and host 3 (slaves). All software components were installed in the three servers: Kafka, Spark, Zookeeper, HDFS, and our own code.

5.1 Dataset characterization

We used two datasets in the experimental evaluation: netflow events from the Los Alamos National Laboratory (LANL) corporate network [25,26], and netflow events that we obtained at a large military infrastructure. The information about the datasets is summarized in Table 2.

Table 3. Attacks in the military network dataset

SrcIP	DstIP	Attack
S1	D1	stealth/slow port scan: 1-to-1 - every port (5sec. pace for 1 day)
S2	/24 net	stealth/slow port scan: one to many - every port (1sec. pace for 1 day)
S3	D3	stealth dictionary attack: SSH auth requests (with 2 min. interval for half day)
S4	D4	stealth dictionary attack: SSH auth requests (with 30 sec. interval for 20 min.)
S5	D5	data exfiltration: unusual volume of data sent to one entity (1.5GB in 7 min.)

LANL dataset This dataset represents 58 consecutive days of pseudonymized event data collected from five sources: authentication events, hosts process start/stop, DNS, netflow, and red team events. We did not use the whole dataset but only the redteam and netflow data. The netflow events have 1.1 GB when compressed and correspond to 129,977,412 events for 12,027 computers. The red team events provide us with attacker IP addresses, only 4, that we use to identify malicious events in the other dataset, i.e., to obtain ground truth for the evaluation. The dataset comes in text files. Each line of the netflow event files contains a timestamp (an epoch time starting at 0), connection duration, source computer, source port, destination computer, destination port, protocol, packet count, and byte count. The well-known ports (e.g., 80 and 443) are not pseudonymized, only the IP addresses. A few sample lines of data are:

```
1,9,C3090,N10471,C3420,N46,6,3,144
1,9,C3538,N2600,C3371,N46,6,3,144
2,0,C4316,N10199,C5030,443,6,2,92
```

Military network dataset This dataset was obtained from the Security Information and Event Management (SIEM) system [6] in production in that network, which collects Netflow events from internal routers. Collecting these flows can give us insights of eventual misbehavior of internal entities, undetected by deployed security systems. The dataset corresponds to a full month, with approximately 5,500 computers and 160 GB of size. As shown in Table 3, we emulated 4 stealth/slow attacks (e.g., probing) at different pace to evaluate OutGene, and provide us with detailed ground truth. We also emulated a noisy attack (high volume of data exfiltrated to an unexpected destination). The main reasons for the chosen emulated attacks were: (1) to be able to evaluate our time stretching analysis; and (2) to have attacks that are unnoticed by traditional protection systems.

5.2 Detection with genetic zoom

This section evaluates OutGene's genetic zoom mechanism. As mentioned before, genetic zoom allows the analyst to understand what are the relevant features for OutGene's decisions (i.e., attack detection after initial clustering). We considered a 24h subset of the LANL dataset in which the attacker with pseudonym IP address C17693 generated many events. We designate the subset *attday*. Notice that no data is available about what attacks were performed by the attacker.

Table 4. Clustering results for day *attday* (LANL dataset)

Initial clustering															
Cluster number	0	10	1	12	9	5	7	3	6	13	4	11	14	2	8
Number of entities	7078	660	12	10	9	6	4	2	2	2	2	1	1	1	1
Clustering after genetic zoom															
Cluster number	0	7	14	5	3	1	11	2	13	10	6	9	12	8	4
Number of entities	7080	646	24	14	7	6	4	2	2	1	1	1	1	1	1

Table 5. Output of genetic algorithm (LANL dataset)

```
-- End of (successful) evolution --
Best allOfFame individual is [0,1,1,0,1,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0], (0.9309)
```

Table 4 shows the clusters obtained for *attday*. We considered the number of clusters $\mathcal{K} = 15$. This value should not be too large or several clusters would represent similar behavior, or too small or there would be no small clusters with outliers. In practice, experience is needed to define the value and obtain good results, but values in the range of 15 to 20 tend to provide good results. As we expected, there are large clusters (2 in this case), and small clusters (13), as observed in the table. Our focus is on clusters having 1 entity. Typically, small clusters correspond either to machines with different yet legitimate behaviour (e.g., web server, web proxy, etc.) or to misbehaving entities. The other clusters might be interesting but without contextual information it is difficult to draw conclusions.

The output of the genetic algorithm (Table 5) shows that there were 8 important features: number of different ports used, number of different ports contacted, sum of packets sent to port 80, sum of packets sent to port 194, sum of packets sent from port 22, sum of packets received, sum of packets received to port 80, and sum of packets received form port 22. Hence, all the oultiers obtained in the initial clustering are also obtained by redoing the clustering with this subset of features (bottom of the table). The results are 93% similar.

To help understanding the meaning of the small clusters, Figure 7 shows a heatmap for the clustering after genetic zoom was applied. The 8 features are at the bottom of the figure (x-axis) and the 15 clusters on the left (y-axis), whereas the color represents the value of each feature for each cluster. In the figure it is possible to observe that the outliers are those with higher values in certain features. IP C17693 (cluster 4) is the known attacker, who contacted many different ports (possibly doing a port scan) and was the only IP contacting port 194. The IP C706 (cluster 10) received several packets to port 80 and contacted several ports, which indicates it corresponds to a webserver, so it is normal behavior. The IP C2091 (cluster 9), also received several packets to port 80, but unlike the previous, it has not contacted several ports, which seems suspicious. The IP C22226 (cluster 6) have high number of packets sent to port 80 which seems suspicious. The IP C5696 (cluster 12) sent several packets from port 22, which may indicate it is a server suffering an attack, or is just one server being accessed by network administrator. The IP C15733 (cluster 8) has several packets received from port 22, which may indicate it is accessing IP C5696.

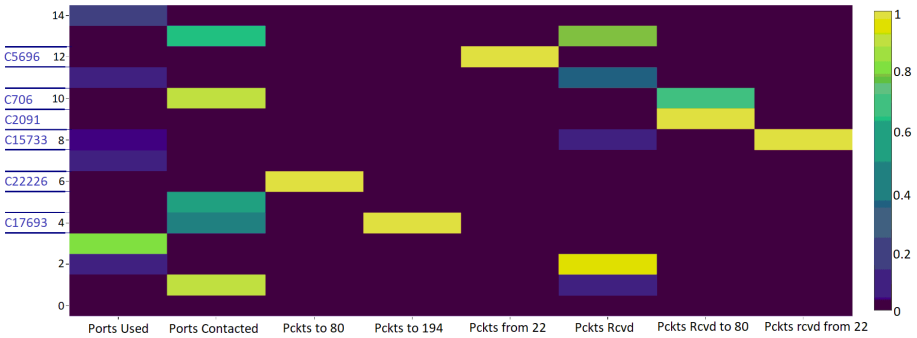

Fig. 7. Heatmap of features vs. clusters after genetic zoom at *attday*, with clusters with a single IP address identified on the left (LANL dataset)

We did a similar exercise with the military network dataset and the 5 attacks we injected (Table 3). The main features identified for each attack were:

- *slow port scan 1-to-1:* SPcon, SP194t, SP25t
- *slow port scan 1-to-/24-net:* SPcon, SP194t, SP25t
- *stealth dictionary attack SSH 2 min.:* SP22t
- *stealth dictionary attack SSH 30 sec.:* SP22t
- *data exfiltration:* SBytes

In summary, instead of having to extract information from the values of all the 26 features, by using the genetic zoom mechanism the analyst understands which subset of features is relevant to differentiate the clusters. In the LANL dataset case, we have limited knowledge about the computers, but we could still extract some information by inspecting the values of these 8 features and identify the red team IP address. That IP was isolated in a cluster, contacted many ports and sent several packets to port 194. In the military network dataset we have more information about both the attacks and the machines, so we can conclude that the mechanism is indeed useful.

5.3 Detection with time stretching

To evaluate time stretching, we used the military dataset with the attacks we injected (Table 3). We ran OutGene and compared results on time-windows from 10 min. to 24 hours, as shown in Table 6. The table is divided in two parts, which show the results in the detection of either attackers or victims. To allow the visualization of the time stretching capabilities, the results are shown for each time-window. Also, clusters are indicated by cluster ranking (the smallest cluster has the highest rank, 1st) and the cluster size in percentage (entities within cluster divided by the total number of entities). Moreover, for each attack, we bolded the time-window were the attack was better detected, i.e., more clearly observable. Next, we provide an interpretation of the table's content.

Table 6. Time stretching evaluation (military network dataset)

Attack detected	cluster rank / % machines in cluster for each time-window 10 min	1h	4h	8h	1day	Comments
Slow port scan 1-to-1 5s	1^{st}/**0.05%**	1^{st}/**0.04%**	1^{st}/**0.04%**	1^{st}/**0.03%**	1^{st}/**0.02%**	Attacker was detected in every window
Slow port scan 1-to-/24-net 1s	1^{st}/**0.09%**	1^{st}/**0.07%**	1^{st}/**0.06%**	1^{st}/**0.05%**	1^{st}/**0.03%**	Attacker was detected in every window
Stealth dict. attack SSH – 2min	9^{th}/30%	7^{th}/8%	1^{st}/**0.04%**	1^{st}/**0.03%**	1^{st}/**0.02%**	Attacker detected in bigger windows
Stealth dict. attack SSH – 30sec	1^{st}/**0.05%**	1^{st}/**0.04%**	1^{st}/**0.03%**	1^{st}/**0.03%**	8^{th}/4.1%	Attacker detected in smaller windows
Data exfiltration out of office time	1^{st}/**0.12%**	4^{th}/0.35%	6^{th}/1.44%	9^{th}/6.4%	11^{th}/7.5%	Attacker in smaller clusters for smaller windows

Detected victim of	cluster rank / % machines in cluster for each time-window 10 min	1h	4h	8h	1day	Comments
Slow port scan 1-to-1 5s	1^{st}/**0.05%**	1^{st}/**0.04%**	1^{st}/**0.04%**	1^{st}/**0.03%**	1^{st}/**0.02%**	Victim was detected in every window
Slow port scan 1-to-/24-net 1s	–	–	–	6^{th}/2%	5^{th}/1.7%	Victims in /24 net grouped in same cluster
Stealth dict. attack SSH – 2min	7^{th}/0.42%	6^{th}/0.4%	5^{th}/0.34%	2^{th}/0.04%	1^{th}/**0.02%**	Victim is a server that changes cluster
Stealth dict. attack SSH – 30sec	1^{st}/**0.8%**	1^{st}/**0.04%**	6^{nd}/0.43%	6^{th}/0.20%	4^{th}/13%	Victim is a server that changes cluster
Data exfiltration out of office time	1^{st}/**0.12%**	3^{th}/0.28%	4^{th}/0.5%	6^{th}/1.8%	7^{th}/4.1%	Victim in smaller clusters for smaller windows

Regarding attack detection, in the case of both *slow port scans* running for almost a day, OutGene achieved excellent results in every time windows. The attacker was completely isolated in every case. Concerning the detection of the *stealth dictionary attack* to the SSH service of half a day and 2 min. intervals, one can observe that the attack is detected (i.e., appears in the highest ranks) starting only from the 4 hour time-window, which is expectable as this is the slowest attack. On the contrary, for a similar attack with a highest pace (30 sec.), the best result is achieved in smaller time windows (starting from 10 min.). In both cases we verified that the suspicious entity is indeed an attacker, not a normal computer with different behavior (e.g., a server). Similarly, for the *data exfiltration attack* where an attacker collects data into a staging point of the internal network, the best results are obtained in the smallest time-window, as the attack took only a few minutes. To conclude, concerning attacker detection, we remark that when analysing time-windows without any trace of emulated attacks, all the machines used to emulate attacks were not in suspicions clusters (e.g., were in the larger cluster).

In relation to victim detection, in the case of *slow port scan* 1-to-1, we can observe that the victim is always in the top ranked clusters. Regarding the slow port scan (1-to-/24-net), there are several victims, as they are all the computers in a /24 subnet. With the smaller time-windows they are scattered in several clusters, but for the larger (8 hours, 1 day) they become more concentrated in smaller, higher rank, clusters (6^{th}/2% and 5^{th}/1.7%). Concerning the *stealth*

Table 7. Performance evaluation (military network dataset)

Attack	Num. alerts	Best time-window	TPR	ACC	FPR (%)
Slow port scan 1-to-1 5s	3	1 day	1.00	0.99	0.04
Slow port scan 1-to-/24-net 1s	4	1 day	1.00	0.99	0.10
Stealth dict. attack SSH 2min	4	1 day	1.00	0.99	0.10
Stealth dict. attack SSH 30sec	3	10min	1.00	0.99	0.07
Data exfiltration	3	10min	1.00	0.98	0.07
Clean traffic	1				0.03

dictionary attacks, having contextual information, we know the victim is a server and usually belongs to small clusters. The only suspicion is the fact that the server moves between clusters and gets in top ranked clusters. Finally, concerning *data exfiltration*, we observe exactly the same behaviour as explained for attack detection (previous paragraph). Besides these emulated attacks, some machines with suspicious behavior have been found and their IPs given to the security team for further investigation.

These results can render values for recall/true positive rate (TPR), accuracy (ACC), and fall-out/false positive rate (FPR), shown in Table 7. Given the usual definitions of true/false positive (right/wrong alarms, TP/FP) and true/false negative (right/wrong no-alarms, TN/FN), we have the usual definitions for the three metrics: TPR = TP/(TP+FN); ACC = (TP+TN)/(TP+TN+FP+FN); FPR = FP/(TN+FP). We count as an alarm a cluster with a single IP. All time-windows in the table include attacks and are assumed to include a single attack each. On average each time-window had 2880 entities. The columns are self explanatory and the main results are on the three columns on the right. As we can observe, the TPR and accuracy are quite high, whereas the FPR is low as desirable (all metrics have values between 0 and 1). The bottom row shows that in (almost) all the windows there was a false alarm, i.e., one cluster with a single computer (a server).

In summary, we can conclude that the time stretching mechanism allows detecting attacks independently of their pace.

6 Related Work

Several surveys offer an extensive review regarding the use of ML techniques in the cybersecurity domain [4,5,20,58]. The first work in this category is apparently due to Lee and Stolfo, who used a supervised ML scheme to detect attacks [29]. Most research in the area used datasets that date back to 1999 and that do not represent the actual cybersecurity landscape nor the difficulty in processing noisy data of real world systems [30, 39, 54, 57].

Most intrusion detection research focus on *signature-based* detection to detect known attacks, or *anomaly-based* detection that typically learns a model of what is considered normal and detects deviations from that model. The first is unable to detect unknown attacks; the former can detect them but at the cost of high false positive rates. OutGene does not need knowledge about what is good or bad behavior, although it assumes that attacks are rare and exhibit distinctive

behavior. Hence, when analyzing previous work, we focused our research on experiments using unsupervised learning methods and using real data at large-scale.

Yen et al. [56] proposed Beehive, one of the first systems exploring the challenges of big data security analytics for real-world log data, using clustering to identify outliers. The main drawback is that processing is done on a daily basis and there is no feature selection method for interpretability. Gonçalves et al. [18] and Sacramento et al. [40] proposed semi-automatic cluster labeling and the implementation of feature extraction using MapReduce framework. The relevant limitations are the same as Beehive's. Veeramachaneni et al. [51] use an ensemble of outlier detection methods and introduces log stream ingestion using window aggregation for efficient feature extraction, although they focus on web logs. The problem of outlier explanation remains. A few other works use related approaches [7, 33, 37, 47, 55].

The previous works rely on batch and mini-batch processing, hence, it is worth mentioning some recent works that do stream analysis, although they rely on knowledge of what good behavior is: Cinque et al. use entropy to infer deviations from a baseline [9]; DeepLog is inspired in natural language processing and interprets logs as elements of a sequence that follows grammar rules [14]; Kitsune uses an ensemble of neural networks called autoencoders to collectively differentiate between normal and abnormal traffic patterns [36].

In what concerns feature selection for clustering, [15] and [3] provide a good overview of several methods. Our choice for the use of a genetic algorithm was inspired in [35, 46], as well as in its success in finance [19] or in simulating attackers' efforts to evade classifiers [53].

Although OutGene was evaluated with examples of SSH brute forcing, port scan and data exfiltration attacks to illustrate the capabilities offered by time stretching, it was not designed to detect specific attacks. However, we mention some works regarding stealth/slow attacks detection. In what concerns detection of SSH dictionary attacks [48, 49] or data exfiltration [33], most works rely on count based features (e.g., count of auth failures). In every case, it is necessary to define thresholds/baselines that can be circumvented. For instance, the popular tool Fail2ban [2], with the default configuration, bans IPs that make 5 failed login attempts over a 10 minutes period. Hence, a smart attacker would never exceed 4 login attempts in each 10 minutes period. Furthermore, he could use different source IPs to perform the attack. Satoh et al. [41] highlight the existence of atypical inter-arrival times between an auth-packet and the next. This feature can be derived from single flows of the SSH protocol. Since this is also a misuse-based method, a malicious user can circumvent it emulating a normal connection by manipulating the sending rate of the packets.

None of the related works in the literature provides mechanisms similar or equivalent to genetic zoom and time stretching.

7 Conclusion

We developed OutGene, an unsupervised learning approach for network intrusion detection, that detects attacks that are undefined (no signatures) without clean training data. It performs clustering of hosts with similar behavior and detects outliers, with the assumption that attackers behave differently from the majority. OutGene advances the state of the art with the genetic zoom and time stretching mechanisms. We show that it is useful to detect attacks in real network data.

References

1. Apache Spark documentation. https://spark.apache.org/, accessed 2019-04-22
2. Fail2ban. https://www.fail2ban.org, accessed: 2019-04-22
3. Alelyani, S., Tang, J., Liu, H.: Feature selection for clustering: A review. Data Clustering **29**, 110–121 (2013)
4. Bhuyan, M., Bhattacharyya, D., Kalita, J.: Network anomaly detection: Methods, systems and tools. IEEE Communications Surveys Tutorials **16**(1), 303–336 (First Quarter 2014)
5. Buczak, A., Guven, E.: A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys Tutorials **18**(2), 1153–1176 (Second Quarter 2016)
6. Cárdenas, A., Manadhata, P., Rajan, S.: Big data analytics for security intelligence. Cloud Security Alliance (2013)
7. Casas, P., Mazel, J., Owezarski, P.: Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. Computer Communications **35**(7), 772–783 (2012)
8. CheckPoint: 2018 security report: Welcome to the future of cyber security (2018)
9. Cinque, M., Corte, R.D., Pecchia, A.: Entropy-based security analytics: Measurements from a critical information system. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 379–390 (Jun 2017)
10. Claise, B.: Cisco systems netflow services export version 9. Tech. rep., RFC 3954 (2004), IETF RFC 3954
11. Debar, H., Dacier, M., Wespi, A.: Towards a taxonomy of intrusion detection systems. Computer Networks **31**(8), 805–822 (Apr 1999)
12. Denning, D.E., Neumann, P.G.: Requirements and model for IDES: A real-time intrusion detection expert system. Tech. rep., Computer Science Laboratory, SRI International, Menlo Park, CA (1985)
13. Dias, L.F., Correia, M.: Big data analytics for intrusion detection: An overview. In: Handbook of Research on Machine and Deep Learning Applications for Cyber Security, pp. 292–316. IGI Global (2020)
14. Du, M., Li, F., Zheng, G., Srikumar, V.: DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In: ACM SIGSAC Conference on Computer and Communications Security (2017)
15. Dy, J.G., Brodley, C.E.: Feature selection for unsupervised learning. Journal of Machine Learning Research **5**(Aug), 845–889 (2004)
16. Fortin, F.A., Rainville, F.M.D., Gardner, M.A., Parizeau, M., Gagné, C.: Deap: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**(Jul), 2171–2175 (2012)
17. Goldberg, D.E., Holland, J.H.: Genetic algorithms and machine learning. Machine Learning **3**(2), 95–99 (1988)

18. Gonçalves, D., Bota, J., Correia, M.: Big data analytics for detecting host misbehavior in large logs. In: Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (2015)
19. Gorgulho, A., Neves, R., Horta, N.: Applying a GA kernel on optimizing technical analysis rules for stock picking and portfolio composition. Expert systems with Applications **38**(11), 14072–14085 (2011)
20. Habeeb, R.A.A., Nasaruddin, F., Gani, A., Hashem, I.A.T., Ahmed, E., Imran, M.: Real-time big data processing for anomaly detection: A survey. International Journal of Information Management (2018)
21. Hellemons, L., Hendriks, L., Hofstede, R., Sperotto, A., Sadre, R., Pras, A.: SSHCure: a flow-based SSH intrusion detection system. In: IFIP International Conference on Autonomous Infrastructure, Management and Security. pp. 86–97. Springer (2012)
22. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification **2**(1), 193–218 (1985)
23. Hunt, P., Konar, M., Junqueira, F., Reed, B.: Zookeeper: Wait-free coordination for internet-scale systems. In: USENIX Annual Technical Conference (2010)
24. Jin, C., Carbonell, J.: Incremental aggregation on multiple continuous queries. In: International Symposium on Methodologies for Intelligent Systems. pp. 167–177. Springer (2006)
25. Kent, A.D.: Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory (2015)
26. Kent, A.D.: Cyber security data sources for dynamic network research. Dynamic Networks and Cyber-Security **1**, 37–65 (2016)
27. Kienzler, R.: Mastering Apache Spark 2.x: Scalable analytics faster than ever. Packt Publishing (2017)
28. Kreps, J., Narkhede, N., Rao, J., et al.: Kafka: A distributed messaging system for log processing. In: Proceedings of NetDB. pp. 1–7 (2011)
29. Lee, W., Stolfo, S.: Data mining approaches for intrusion detection. In: Proceedings of the 7th USENIX Security Symposium (Jan 1998)
30. Leung, K., Leckie, C.: Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the 28th Australasian Conference on Computer Science. pp. 333–342 (2005)
31. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. pp. 281–297 (1967)
32. Mandiant: Special report, M-TRENDS 2018 (2018)
33. Marchetti, M., Pierazzi, F., Colajanni, M., Guido, A.: Analysis of high volumes of network traffic for advanced persistent threat detection. Computer Networks **109**, 127–141 (2016)
34. Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al.: MLlib: Machine learning in apache spark. Journal of Machine Learning Research **17**(1), 1235–1241 (2016)
35. Middlemiss, M., Dick, G.: Feature selection of intrusion detection data using a hybrid genetic algorithm/knn approach. In: Design and application of hybrid intelligent systems. pp. 519–527. IOS Press (2003)
36. Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, A.: Kitsune: An ensemble of autoencoders for online network intrusion detection. In: Proceedings of the Network and Distributed System Security Symposium (2018)
37. Osada, G., Omote, K., Nishide, T.: Network intrusion detection based on semi-supervised variational auto-encoder. In: European Symposium on Research in Computer Security. pp. 344–361 (2017)
38. OTA: Cyber incident & breach trends report (2018)
39. Otey, M.E., Ghoting, A., Parthasarathy, S.: Fast distributed outlier detection in mixed-attribute data sets. Data Mining and Knowledge Discovery **12**(2-3), 203–228 (2006)

40. Sacramento, L., Medeiros, I., Bota, J., Correia, M.: Flowhacker: Detecting unknown network attacks in big traffic data using network flows. In: 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. pp. 567–572 (2018)
41. Satoh, A., Nakamura, Y., Ikenaga, T.: A flow-based detection method for stealthy dictionary attacks against secure shell. Journal of Information Security and Applications **21**, 31–41 (2015)
42. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: IEEE 26th Symposium on Mass Storage Systems and Technologies. pp. 1–10 (2010)
43. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: Proceedings of the 30th IEEE Symposium on Security and Privacy. pp. 305–316 (2010)
44. Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., Stiller, B.: An overview of IP flow-based intrusion detection. IEEE Communications Surveys and Tutorials **12**(3), 343–356 (2010)
45. Srisuresh, P., Holdrege, M.: IP network address translator (NAT) terminology and considerations. IETF Request for Comments: RFC 2663 (Aug 1999)
46. Stein, G., Chen, B., Wu, A.S., Hua, K.A.: Decision tree classifier for network intrusion detection with GA-based feature selection. In: Proceedings of the 43rd ACM Annual Southeast Regional Conference, Volume 2. pp. 136–141 (2005)
47. Stergiopoulos, G., Talavari, A., Bitsikas, E., Gritzalis, D.: Automatic detection of various malicious traffic using side channel features on TCP packets. In: European Symposium on Research in Computer Security. pp. 346–362 (2018)
48. Su, Y.N., Chung, G.H., Wu, B.J.: Developing the upgrade detection and defense system of SSH dictionary-attack for multi-platform environment. iBusiness **3**(01), 65 (2011)
49. Thames, J.L., Abler, R., Keeling, D.: A distributed active response architecture for preventing SSH dictionary attacks. In: IEEE Southeastcon. pp. 84–89 (2008)
50. Turcotte, M.J.M., Kent, A.D., Hash, C.: Unified Host and Network Data Set, chap. 1, pp. 1–22 (Nov 2018)
51. Veeramachaneni, K., Arnaldo, I., Cuesta-Infante, A., Korrapati, V., Bassias, C., Li, K.: AI^2: Training a big data machine to defend. In: Proceedings of the 2nd IEEE International Conference on Big Data Security on Cloud (2016)
52. Whitley, D.: The GENITOR algorithm and selection pressure. In: Proceedings of the 3rd International Conference on Genetic Algorithms. pp. 116–121 (1989)
53. Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers. In: Proceedings of the 2016 Network and Distributed Systems Symposium (2016)
54. Yamanishi, K., Takeuchi, J.I., Williams, G., Milne, P.: On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. Data Mining and Knowledge Discovery **8**(3), 275–300 (2004)
55. Yen, T.F.: Detecting stealthy malware using behavioral features in network traffic. Ph.D. thesis, Carnegie Mellon University Department of Electrical and Computer Engineering (2011)
56. Yen, T.F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., Kirda, E.: Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. In: Proceedings of the 29th ACM Annual Computer Security Applications Conference (2013)
57. Zhang, J., Zulkernine, M.: Anomaly based network intrusion detection with unsupervised outlier detection. In: 2006 IEEE International Conference on Communications. vol. 5, pp. 2388–2393 (2006)
58. Zuech, R., Khoshgofthaar, T., Wald, R.: Intrusion detection and big heterogeneous data: a survey. Journal of Big Data pp. 90–107 (2015)