Search-Based Test and Improvement of Machine-Learning-Based Anomaly Detection Systems

Maxime Cordy
maxime.cordy@uni.lu
SnT, University of Luxembourg
Luxembourg

Steve Muller
smuller-research@outlook.com
Luxembourg

Mike Papadakis
mike.papadakis@uni.lu
SnT, University of Luxembourg
Luxembourg

Yves Le Traon
yves.letraon@uni.lu
SnT, University of Luxembourg
Luxembourg

ABSTRACT

Machine-learning-based anomaly detection systems can be vulnerable to new kinds of deceptions, known as training attacks, which exploit the live learning mechanism of these systems by progressively injecting small portions of abnormal data. The injected data seamlessly swift the learned states to a point where harmful data can pass unnoticed. We focus on the systematic testing of these attacks in the context of intrusion detection systems (IDS). We propose a search-based approach to test IDS by making training attacks. Going a step further, we also propose searching for countermeasures, learning from the successful attacks and thereby increasing the resilience of the tested IDS. We evaluate our approach on a denial-of-service attack detection scenario and a dataset recording the network traffic of a real-world system. Our experiments show that our search-based attack scheme generates successful attacks bypassing the current state-of-the-art defences. We also show that our approach is capable of generating attack patterns for all configuration states of the studied IDS and that it is capable of providing appropriate countermeasures. By co-evolving our attack and defence mechanisms we succeeded at improving the defence of the IDS under test by making it resilient to 49 out of 50 independently generated attacks.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Software and its engineering → Software testing and debugging**.

KEYWORDS

software testing, intrusion detection systems, clustering

ACM Reference Format:
Maxime Cordy, Steve Muller, Mike Papadakis, and Yves Le Traon. 2019. Search-Based Test and Improvement of Machine-Learning-Based Anomaly Detection Systems. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3293882.3330580

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07…$15.00
https://doi.org/10.1145/3293882.3330580

1 INTRODUCTION

Modern networks and systems evolve at a rapid pace, both in terms of numbers and behaviour. To support this polymorphic evolution, new-generation anomaly detection systems employ machine learning to provide adaptive and robust security mechanisms.

To this end, learning-based Intrusion Detection Systems (IDS) [10], a particular class of anomaly detection systems that aims at detecting security breaches within a network or a system, constantly adapt to system and network evolutions. Their key attribute is that they learn to recognize and adapt to the typical behaviour of systems and thus are able to detect suspicious and anomalous deviations that result from malicious attacks [6]. As such, those IDS provide security mechanisms that are immediately applicable to production environments and remain adaptive to system and environmental evolutions (such as behaviour shifts, appearances and disappearances). These characteristics are appealing and preferable over the traditional ones [6].

Among the related machine learning approaches (e.g. neural networks), those based on unsupervised clustering are the most prominent [17]. They offer comprehensive and explainable results, they do not require any offline training or prior knowledge, and unlike, e.g., Bayesian approaches, they can model multiple types of behaviours at the same time.

Unfortunately, machine-learning-based IDS are vulnerable to new types of attacks, called training attacks [11]. These attacks exploit the live learning mechanism of the systems by progressively injecting small portions of abnormal data. Although the injected data are harmless, they seamlessly swift the learned states of the systems to a point where illicit data can pass unnoticed [18].

Clustering-based IDS are no exception [4], as attackers can slowly modify the numbers and shapes of the clusters. Previous work has shown that a clustering-based IDS monitoring network traffic against Denial-of-Service (DoS) attacks can be deceived [11]. To counterbalance this problem, state-of-the-art countermeasures run multiple instances of clustering algorithms with different parameters to monitor the traffic in parallel. Then, an alert is raised as soon as one of the instances suspects that an attack occurs.

Such a solution can succeed in detecting the training attack but offers no confidence that it is resilient to attacks performed differently, e.g., a DoS attack with a different speed. Therefore, there is an emerging need to practically assess the resilience of such systems. Naturally, this can be achieved through testing, where one designs tests exhibiting different training attack schemes.

However, as in every testing scenario, test design is hard and time-consuming. This activity is usually performed manually and in an ad-hoc way. Therefore, administrators rely on the knowledge and ability of testers to assess their defence mechanisms.

In an endeavour to address this challenge, we propose a systematic testing approach for clustering-based IDS to counter training attacks (the threat model). Our method realizes the idea that one can make IDS more resilient to real-world, black-box training attacks, by confronting them to white-box training attacks crafted by observing IDS cluster evolution.

We formulate the crafting of white-box training attacks as a search problem and shows that search-based techniques can generate effective attack schemes. We also propose the inverse technique, i.e., a search technique that searches for countermeasures (defence strategies) to counter given attacks. Going a step further, we co-evolve both the attack and defence schemes in order to overall achieve better resilience and stronger defences.

We implemented this approach and performed a case study on a denial-of-service attack detection scenario using network traffic data recorded from a real-world system. Our results show that our framework can systematically generate attack schemes bypassing the current state-of-the-art defences, i.e., multiple clustering instances [11]. We also show that for any successful attack our framework can generate an appropriate defence without disturbing the normal (benign) traffic.

Overall, since our two methods (automatic generation of attacks and defences) are complementary, they can be applied iteratively. Thus, by co-evolving both attacks and defences, one can reliably improve the resilience and robustness of the systems under analysis.

In summary, the paper makes the following contributions:

(1) We design a testing framework based on search algorithms that, given an IDS, can generate a test (i.e. a training attack scheme) to deceive the IDS. We instantiate this framework in a denial-of-service attack scenario, where a training attack consists of unnoticed, successive increases in data rate.
(2) We generalize the countermeasure proposed by Muller et al. [11] and design a genetic algorithm to search for a defence strategy (in the form of new parameter sets) that succeeds in detecting the attack induced by a given test case.
(3) We show that our testing framework enables the continuous improvement of IDS detection capabilities. More precisely, we set up a co-evolution process that attractively generates attacks and defences leading to new parameter sets that make the IDS resilient to all generated attacks.

2 BACKGROUND AND CASE STUDY

2.1 Clustering-Based IDS

D-Stream [7] is a clustering algorithm that can categorize n-dimensional data streams in real time. Its principles are illustrated in Figure 1. D-Stream divides the data state space into a predefined grid

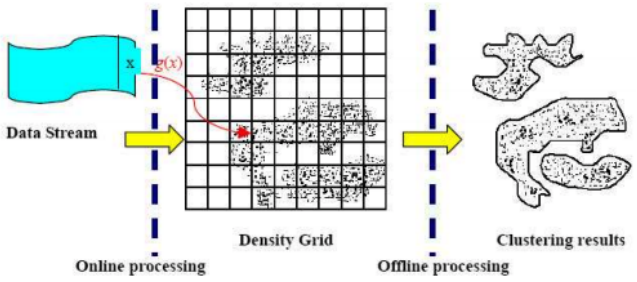

Figure 1: D-Stream online and offline processes [7].

and records new streamed data as data points in the corresponding grid cells. Newer data are considered more important than older ones, as they better reflect the current behaviour of the system. Therefore, D-stream associates each recorded data point with a weight that decays over time. The rate of decaying is controlled by a meta-parameter, denoted by λ, with $0 < \lambda < 1$, such that the weight of a data point at time t is given by λ^{t-t_0} where t_0 is the time at which the data point was recorded. Thus, the higher λ is, the longer the system remembers past data.

A cluster is defined as a maximal contiguous set of dense cells. A cell is dense at a given time if and only if the sum of the weights of its contained data points is higher than a specified threshold. This notion of density allows one to distinguish recurrent behaviour (represented by dense cells) from outliers (represented by isolated data points in sparse cells, which are thus not part of any cluster).

Outliers corresponds to statistical abnormalities that do not reflect changes in the system behaviour. As such, they are considered unharmful [11]. On the contrary, the formation of clusters at unexpected spaces of the grid may result from threatening persistent attacks. Therefore, any modifications to the shape of the clusters should raise attention, as those are signs of persistent modifications in the behaviour accepted by the system.

The shape of clusters can change in multiple ways. When an isolated cell becomes dense, it gives rise to a new cluster. This phenomenon models the capture of a new type of behaviour by the clustering algorithm. Clusters can also disappear when their constituent cells lose their dense status (i.e. due to decayed data points). This happens when a previously recurrent behaviour is now considered as an outlier. More generally, clusters can evolve (i.e. gain or lose cells) to give account for natural evolutions of a modelled behaviour (e.g. more monitored smart equipments yields an increase of network traffic). Two particular cases of evolution are the merging and the splitting of clusters. Two disjoint clusters merge as soon as they become connected via a new dense cell. This phenomenon originates from the fact that two independent behaviours have evolved so much that they are now very similar. Similarly, one cluster can split due to a natural separation of its modelled behaviour into new classes.

An IDS based on D-Stream consists of two processes (see Figure 1). First, it feeds the grid with the received data in an online process. Second, after a specified interval of time Γ, it triggers and offline clustering process and observes the changes in the clusters' shape. Only the appearance of new clusters is considered harmful,

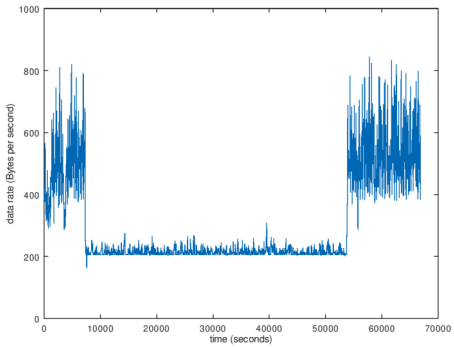

Figure 2: The normal traffic rate (in Bytes per second) of our SCADA network case study, recorded for 18 hours.

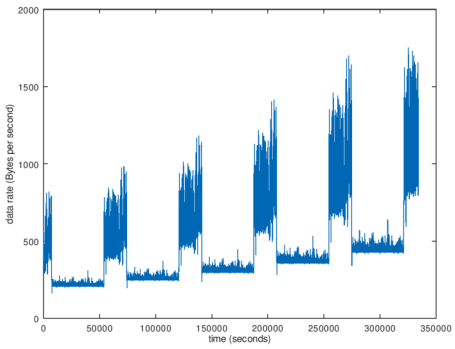

Figure 3: The traffic rate (in Bytes per second) of our SCADA network case study confronted to a training attack.

as it corresponds to recurrent behaviour that was before rarely observed. This anomaly is thus assimilated to an attack.

Training attack refers to a sophisticated attack scheme that hijacks the clustering process to make inappropriate behaviour accepted by the IDS. It progressively injects malicious data in order to move one or more clusters so slowly that these changes are perceived as licit evolutions. Intuitively, the trick lies in the frequency of observations performed by the IDS: frequent observations make the attack look like a succession of small, unharmful changes in the shape of the clusters, while longer-term observations reveal the appearance of a new cluster. Longer-term observations alone, however, are not viable, as they are vulnerable to short-term attacks.

To counter short-term *and* long-term attacks, the state-of-the-art countermeasure [11] recommends to run multiple instances of D-stream in parallel, each of which works with a different Γ interval. Thereby, one increases the likelihood that at least one instance will observe the appearance of new clusters, thus detecting training attacks. This, of course, comes at the cost of a multiplication of the required resources by a factor equal to the number of running instances. It is therefore essential to select the Γ values wisely, so as to minimize the number of instances while maximizing the likelihood of detecting training attacks.

An alternative solution is to slow down the training process [4], but doing so runs the risk of rendering the IDS inert. Therefore, the only countermeasure we use throughout our work is the addition and configuration of new D-Stream instances (along the lines proposed by Muller et al. [11]).

2.2 Case Study: DoS Attack on SCADA Network

We consider a specific case study that serves as a running example throughout the paper and as an evaluation case for our experiments. This case concerns a clustering-based network monitoring system that targets Denial-of-Service (DoS) attacks. Such attacks consist of flooding a network with data packets in order to overload the supporting infrastructure. The protected network we consider is a Supervisory Control and Data Acquisition (SCADA) network, which is commonly deployed to support the operations of critical infrastructures such as, e.g., water treatment facilities, manufacturing processes, airports and space stations.

Dataset [1]. To detect DoS attacks, the monitoring system follows typical recommendations [5] and records the average data rate (in Bytes per second) received by the system every minute. Figure 2 shows the traffic rate of the SCADA network recorded for 18 hours. It illustrates the two behaviours that the network exhibits, i.e. the day traffic and the night traffic. A D-Stream-based IDS would cluster the data rates to reflect the two behaviours. If we assume a linear grid with 200-byte-long cells, it is likely that at least two cells would be dense and form a contiguous cluster: the first ranges from 200 to 400 and contains data points of the night traffic, the other from 400 to 600 and corresponds to the day traffic.

Should the data rate frequently reach unusual values, the system raises an alert. Yet, it is evident that the data rate of a network system is unlikely to remain constant over time. For instance, an increase of licit activities within the network yields a higher data rate. This makes clustering algorithms such as D-Stream particularly appropriate, as those can register natural variations in the data rates without triggering false alarms.

At the same time, however, this also opens the door for training attacks that would progressively increase the data rate over time. This particular attack has already been identified by the study of Muller et al. [11]. In order to generate a training attack, they artificially injected additional data, thereby increasing the data rate following a carefully chosen exponential law. Their training attack was absolutely unnoticed by the IDS.

Figure 3 illustrates the effect of such an attack on the SCADA network. The particular attack increases the data rate by 20% every 18 hours. We see that both day and night traffic rates are slowly increasing. At each 18-hour iteration, the linear grid would progressively expand and/or shift the cluster towards higher cells to account for the relative increase. However, in the long run, we would get two clusters, i.e. the single-cell cluster ranging from 200 to 400 Bps and the other formed by cells from 800 to 1,600 Bps.

With their experiments, Muller et al. showed that a countermeasure to this attack was to use multiple instances of D-Stream, each of which doubles the Γ value of the previous one. Nevertheless, although they demonstrated the possibility of training attacks and countermeasures, they considered only one specific case (settings), evaluated manually both for attack and defence. Therefore, there is no guarantee that IDS are generally vulnerable and that

their solution is secure over systematic attacks or other similar attack schemes. More generally, their focus was to demonstrate the possibility of the attack rather than to provide a systematic testing approach. Therefore, through our work, we aim at laying the foundations to address the automated testing and resilience improvement problem in the context of training attacks.

3 SEARCH-BASED IMPROVEMENT OF IDS

In order to provide an automated and systematic approach to test D-Stream-based IDS, we propose a methodology based on search algorithms. We present it first under the particular scope of the DoS attack case study, before discussing how to generalize the methodology to other cases (including with n-dimensional data).

3.1 Search-Based Test of IDS

We designed a genetic algorithm which, given an IDS running D-Stream, generates attack schemes capable of deceiving the IDS. In what follows, we present its constituents.

Genotype. Each individual of the population generated by our algorithm corresponds to a training attack scheme, i.e. a strategy to make attacks unnoticed. In our case study, the genotype comprises only one chromosome (since we deal with one-dimensional data). The genes of this chromosome reflect the strategy used to increase the data rate.

A first, yet inefficient, representation contains one gene per 60 seconds timeframe (which is also the interval of time between two observations of the network's input data rate), where each gene specifies how many Bytes per second the attacker should send during the corresponding timeframe. Although this solution offers the most flexibility, it requires a lot of genes to represent a single attack (e.g., our experiments consider a total duration of one year, which means that the chromosomes can have up to 525,600 genes). It is therefore computationally expensive.

A second, more efficient, consists of a single gene, which is the exponential rate for increasing the data rate over time. This solution, however, offers less control and therefore appears as less stealthy.

As a compromise between the above two representations, we propose a third solution that offers flexibility and results in chromosomes of affordable size. We specify that a chromosome is composed of k ordered real-valued genes, each of which corresponds to a timeframe whose length is 1/k of the total duration allowed for the attack. Each gene models how much the current data rate should increase during its corresponding timeframe. That is, at any i-th timeframe, $0 < i \leq k$, the data rate received by the system will be equal to $r \times \prod_{j=1}^{i}(1 + ch[j])$ where r is the original data rate received by the system (i.e. in the absence of attack) and $ch[j]$ is the value of the j-th gene of the chromosome.

To improve scalability, we limit the maximum size of the chromosome (i.e. the number of successive raises in data rate). We also limit the maximal value of a gene in order to avoid large fluctuations that would rapidly be detected. In our experiments, we have set the maximum number of variations to 365 (i.e. one per day) and the maximum gene value to 1.0 (i.e. a maximum increase of 100%).

Figure 4 illustrates an attack scheme produced by our testing algorithm. This scheme divides the attack duration into 16 time fragments (shown on the x-axis). For each fragment, the y-axis shows the multiplying factor that is applied on the original data rate (obtained by multiplying the gene values). This attack scheme was successful at deceiving up to 7-instance IDS (see Section 5).

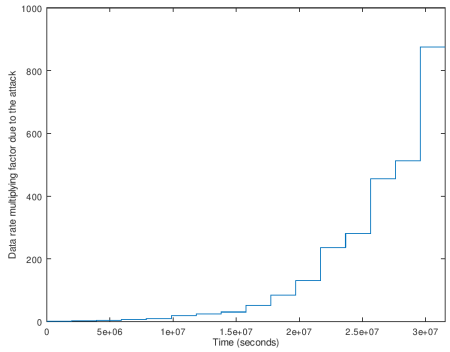

Figure 4: A generated attack deceiving up to 7-instance IDS.

Fitness function. Recall that behaviours accepted by the system are represented by clusters of dense cells in the D-Stream grid. Then, the main goal of an attacker is to make the IDS form a cluster that encompasses data points corresponding to malicious behaviour. We represent this objective as a single data point named $target$, which represents a typical malicious data point that the attacker wants the system to accept. The attack thus aims to change the clusters' shape so that $target$ falls inside one cluster.

To measure the progress towards this objective, we compute the vector distance between $target$ and the closest centroid over all clusters built by the IDS. Intuitively, centroids represent the data points that will the most likely be accepted by the clusters. If some malicious data points are close to a centroid, it means that their recording within the system will contribute to stabilizing the cluster around its centroid, thereby increasing the likelihood that the corresponding illicit behaviour remains accepted in the future. Then, the percentage of progress accomplished by an attack can be measured wrt. (a) the initial distance computed from the location of the cluster centroids before the attack is executed and (b) the remaining distance after executing the attack.

An alternative to centroid distance is the average distance to all data points of the cluster. While this solution seems suitable, it is inherently more expensive to compute and is therefore disregarded.

The second objective of an attacker is to complete the training attack as soon as possible, as he may lose opportunities otherwise. Hence, we define the time consumed by a training attack with respect to a specified maximum time budget, denoted by $duration$.

Thus, the success of a training attack can be measured as a trade-off between maximizing the progress of an attack and minimizing the consumed time budget, all this while avoiding detection. To model this trade-off, we define our fitness function as a linear equation over the percentages of progress and consumed time budget, i.e. $\alpha \times progress\% - \beta \times consumed\%$ where $\alpha, \beta > 0$ are meta-parameters that specify the relative importance of progress and time.

Alternatively, instead of a linear equation, we could have defined two fitness functions, thereby reducing our problem to multi-objective optimization. It is, however, realistic to assume that what

is important for attacks is to succeed within the allowed time budget, and that time should only serve to rank attacks that are successful. In practice, α and β will be set such that progress has substantially more weight than time. In our experiments we set $\alpha = 10$ and $\beta = 1$.

In the case of detected attacks, we want to reward attacks that remained unnoticed for the longest time. Yet, those should never be considered better than undetected attacks. Theoretically, the fitness value of an undetected attack can never go below $-\beta$, as this corresponds to 0% of progress and 100% of consumed time. Thus, the highest fitness value that the best performing detected attack (which is detected at the total duration) can receive is $-\beta$. Thus, our fitness function is defined as follows.

Definition 3.1. Let *ids* be a clustering-based IDS, *atk* an attack scheme run against *ids*. Then the fitness function of *atk* over *ids*, denoted by $fitness(atk, ids)$, is given by

$$-\beta \times \frac{time}{duration}$$

if *ids* detected *atk* on *time* time units. Otherwise, it is given by

$$\alpha \times \frac{distance_0 - distance}{distance_0} - \beta \times \frac{time}{duration}$$

where *distance* (resp. $distance_0$) denotes the distance between *target* and the closest cluster centroid of *ids* after (resp. before) starting the execution of the attack scheme and *time* denotes the time needed to perform the attack.

In our SCADA network traffic example, we assume that attackers perform a DoS attack by increasing the data rate received by the network up to a predefined target rate, which represents the maximal load of the network. Accordingly, *target* is a real value and the distance is measured by the scalar difference between *target* and the closest centroid. Then, before executing an attack, we first run the tested IDS against normal (unaltered) traffic data (our input dataset). We computed the centroid of the two produced clusters, which encompass data rates received by the system during the day and during the night. The distance between the highest cluster centroid and the *target* value, i.e. $distance_0$, is then used as the starting point to measure the progress of an attack towards *target*. The advantage of this modelling is that it is independent of the actual attack scheme. However, it requires the use of some data representing the normal state of the traffic.

Selectors and alterers. At each iteration of the genetic algorithm, a subset of the individuals is selected to pursue evolution, and some of them are being altered to produce new individuals. Generally speaking, an appropriate choice of selection and alteration operators largely depends on the modelled problem. In our experiments, we considered the most typical settings, which were found to perform well. For selection, we use a tournament selector that keeps the best individuals out of samples of three. As for alterers, two individuals are combined to form an offspring based on single-point crossover with a 0.2 recombination probability, while gene mutation in the offspring occurs with a probability of 0.15.

3.2 Search-Based Improvement of IDS

Our next step is to propose countermeasures to improve the IDS in detecting training attacks. To achieve this, we generalize the countermeasure proposed in [11], i.e. running multiple instances of D-Stream with different clustering intervals.

In the following, we describe a second genetic algorithm that can, given an attack scheme, determine how many instances and what clustering intervals are needed to detect attacks executed by this scheme. We name such a solution a *defence strategy*.

Genotype. Each individual of the population represents a defence strategy. A strategy defines a number of D-Stream instances as well as their respective parameters. Given a data space, D-stream stands with the following parameters: the grid parameters (i.e. the number of cells N, their coordinates on the grid, their density threshold C_m, and the data decay rate λ), and the interval of time Γ between two observations of the clusters. Previous work [11] has shown that appropriate values for grid parameters are case-specific and interrelated. Therefore, two strategies are distinguished by the number of parallel clustering and their interval values Γ. Accordingly, we represent the genotype of an IDS with a single, variable-sized chromosome of integer-valued genes. The size m of the chromosome represents the number of D-Stream instances to run, and value of the ith gene is the interval value Γ_i of the ith instance. For example, the chromosome [600, 1200, 2400] represents a defence strategy invoking three instances that observe changes in the clusters every 600, 1200 and 2400 seconds, respectively.

Fitness function. A defence strategy should obviously allow the detection of as many attacks as possible. Conversely, it should not be too restrictive in order to accept normal fluctuations in the system. It is indeed essential to avoid raising false alerts. Therefore, any defence we built is first executed against normal data as a sanity check. Regarding real attacks, a defence should aim at the fastest detection time while, paradoxically, minimizing resources (i.e. the number of deployed instances). Successful attacks that remain undetected should be held off as long as possible.

We propose to score defence strategies through a fitness function to minimize. For detected attacks, this function involves minimizing the percentage of detection time (out of an apriori maximal duration) and of used resources (wrt. a specified maximum number M of instances). For undetected attacks, it attempts to maximize the attack execution time in place of minimizing detection time. As in the previous algorithm, we model the trade-off between time and resource consumption as a linear equation.

Definition 3.2. Let *ids* be a clustering-based IDS and *atk* an attack scheme executed against *ids*. Then the fitness function of *ids* over *atk*, denoted by $fitness(ids, atk)$, is given by

$$\gamma \times \frac{time}{duration} + \delta \times \frac{m}{M}$$

if ids detected *atk*; otherwise, it is given by

$$-\nu \times \frac{time}{duration} + \delta \times \frac{m}{M}$$

where $\gamma, \delta, \nu > 0$ are meta-parameters that specify the relative importance of detection time, resources and attack hold-off time.

In our experiments, we assumed that detection and hold-off times are more important than resource consumption and set the meta-parameters accordingly: $\gamma = 10, \delta = 1, \nu = 10$.

Selectors and alterers. To make the population evolve, we rely on the same selectors and alterers as those used to generate attacks,

i.e. ternary tournament selector, single-point crossover with 0.2 probability, and gene mutation with a 0.15 probability.

3.3 Co-Evolution of Attacks and Defences

When deployed in production, an IDS should be able to detect as many attacks as possible. Thus, our ultimate goal is to create defence strategies that are increasingly resilient to attack schemes without triggering false alarms. To achieve this, we designed a co-evolution process that iteratively generates strengthened defences based on previous, successful attack schemes.

Its inputs are an initial, manually set defence strategy d_0, and a number k of iterations to perform. Each i-th iteration consists of a new co-evolution step. During such step, we first generate a new attack scheme a that deceives the defence d_{i-1}, which was either the defence resulting from the last iteration or, for the first iteration, the initial defence d_0. Then, we generate a new defence d_i that successfully detect a and all the previous attacks. To achieve this, we adapt the genetic algorithm presented in Section 3.2 such that the fitness value of an individual is obtained by summing up the fitness values obtained wrt. to each individual attack schemes.

3.4 Generalization to N-Dimensional Cases

Since our primary contribution regards the formulation and use of metaheuristic search techniques for the test and improvement of IDS, we only provide examples and empirical evidence (see Section 5) that search can beneficial on a 1-dimensional realistic case.

Interestingly, the need for search methods is intensified when adding more dimensions, because the solution space is decreasing while the search space explodes. Of course, in this case, our framework requires some tuning, but even as it is, our fitness functions, co-evolution scheme and, more generally, the whole approach are independent of the domain and support multi-dimensional data.

We can easily adapt our technique on other ML-based IDS. For instance, the two objective functions we use are independent of the domain and the dimensions. They are only concerned with time, centroid distance, and number of D-Stream instances. Of course, with 2 dimensions or more, the scalar distance must be replaced, e.g. by Euclidean distance. The selectors and alterers we chose are those commonly used in genetic algorithms, without any tuning. Thus, they can be applied as they are. D-stream clustering natively supports n-dimensional data and the genotype of our defence strategies only concerns D-stream parameters (number of instances and clustering intervals).

In the end, to apply our framework to another case study we would need to change only the genotype of the white-box training attacks. To do so, we propose a general recipe where the genotype of individuals (attack schemes) consists of n chromosomes, that is, one chromosome per data dimension. Each chromosome actually encodes a strategy to alter a part (i.e. a dimension) of the malicious data. The types and number of the genes composing each chromosome depends on the considered data dimension and strategy; it is thus problem- and solution-specific.

Overall, if we stick to training attacks our method remains general (with minor tweaks). Beyond that, it is a matter of the exact data manipulation and the monitoring that the IDS relies on.

4 RESEARCH QUESTIONS

It is virtually impossible to build an IDS that can defend against any attack. To increase confidence, one can manually design an attack scheme and apply it to the to-be-deployed IDS to assess its detection capabilities. However, coming up with a successful attack can be hard. Therefore, our first questions concern the ability of our search technique to perform successful attacks.

To answer this question we need to distinguish between single and multiple instance IDS. The former case forms the current state of IDS (that are known to be vulnerable [11]), while the later forms the current state-of-the-art countermeasure [11]. Thus we ask the following two subquestions:

RQ1.1. Can we automatically generate successful training attacks for single instance IDS?

RQ1.2. Can we automatically generate successful training attacks for multiple instance IDS?

We seek for a positive answer to the first subquestion in order to show that our method is effective at generating potential training attacks. Such an answer will also re-validate the findings of Muller et al. [11] and necessitate the need for multiple instance IDS. Similarly, answering positively the second subquestion will provide evidence that the existing countermeasure is insufficient and that our method is capable of finding security gaps on state-of-the-art defences.

In practice, there might be cases where countermeasures are not sufficiently configured and trained. In these cases, it should be easy to reconfigure and improve the system defences. However, how this could be performed? This leads to our second research question:

RQ2. Given a training attack, can we automatically find effective defences?

An answer to this question suggests a potential improvement of the system defences. However, how such a reconfiguration perform on other attack schemes? In other words, can the automated defence strategies generalize the improvements and offer higher resilience? This leads us to investigate a third research question:

RQ3. Can we improve the resilience of IDS by making attack and defence schemes co-evolve?

Security testing aims at spotting security gaps and improving countermeasures. To this end, we seek a self-learning (co-evolving) approach that learns to attack and defend with the ultimate goal of improving the security countermeasures given a range of solutions.

5 RESULTS

5.1 Experimental Setup

To address our research questions, we implemented our testing approach in a prototype tool. The tool was developed in Java on top of Jenetics, an established framework for modelling and executing genetic algorithms. Our full implementation is publicly available for replication purposes.[1]

All meta-parameters of the genetic algorithms were left to the default value assigned by Jenetics, except population size. We set the population size to 24 (instead of 50) in order to reduce the required computation time. A higher population size did not appear to affect our conclusions. Also, we stop the evolutionary

[1] https://bitbucket.org/maxcordy/idsga

testing algorithm that generates attack schemes as soon as no better individual has been found for 20 successive generations. As for the defence generation algorithms, we stop it after 50 successive generations without improvement. These numbers were found experimentally to be sufficient to find successful solutions.

All our experiments focus on our case study, i.e. detecting DoS training attacks over a network. To that aim, we consider the 4SICS Geek Lounge dataset [1], which contains 18 hours of real SCADA traffic data. This dataset has the particularity of exhibiting working-hour traffic (roughly 800 bytes per second) and non-working-hour traffic (roughly 300 bytes per second). We repeated the data to reach a total duration of one year, thereby obtaining a more realistic case.

All instances of D-Stream used in our experiments have been assigned the same grid parameters. The grid data space is divided into a logarithmic scale of $N = 50$ cells, ranging from 2^7 to 2^{20} Bytes per second. The decay rate is set to $\lambda = 0.01^{30^{-1}days}$, which makes the D-Stream instances forget data after 30 days. The cell density threshold is set accordingly to 0.1, following the recommendations of [11], which also allows us to replicate their experimental settings.

All D-Stream instances used by defences were first trained on the legit data of the first 18 hours to avoid false positives. To ensure this, we run each instance on one-year-long unmodified traffic data and noticed that no false alert was raised, while reducing the training time actually did raise some alerts. By training for 18 hours, we ensured that all data are recorded once by the D-Stream instances.

Each experiment run invokes one or more randomly-seeded genetic algorithms. To account for random variations, we repeated RQ1's and RQ2's experiments 50 times. Unless noticed otherwise, the results we provide are the median of the individual runs.

Here it must be noted, that the above settings are general ones, common to all RQs we investigate. Still the specific settings required to answer each specific RQ are given at the beginning of the Sections answering them (Sections 5.2, 5.4, 5.5)

5.2 RQ1.1: Searching for Training Attacks on Single-Instance IDS

We first consider 7 different IDS, each of which is composed of a single D-Stream instance. The instance of the first IDS has a clustering interval of 600 seconds, while each successive instance doubles the interval of the previous one. The 7th IDS thus employs a clustering interval of 38,400 seconds (that is, 10 hours and 40 minutes). According to a survey of Incapsula [9], an established company providing solutions to mitigate DoS attacks, 86% of such attacks last less than 24 hours and 68% less than 12 hours. Therefore, further doubling the interval seems irrelevant as this would impede the detection of the majority of attacks before they terminate.

For each of the 7 IDS, we apply our evolutionary testing algorithm (see Section 3) to generate attack schemes that we execute against the considered IDS. To do so, we set the weight parameters $\alpha = 10$ and $\beta = 1$; the minimum and maximum gene values to 0 and 1 respectively, so as to ensure that the data rate never suddenly gets more than twice higher as its current value. The minimum and maximum numbers of genes are set to 1 and 365 respectively, which means that a new increase in data rate occurs at most once per day and that there will be at least one increase throughout the whole duration of the attack.

For all 7 IDS, our algorithm managed to generate an attack scheme that succeeded in moving a cluster centroid to the desired target rate or above. Figure 5 shows, for each IDS (represented by its interval value) and over 50 repeated runs, the time required by the best attack schemes to accomplish their attack. Overall, the vast majority of the best attack times range from 530,000 seconds to 875,000 seconds (i.e. 6 to 10 days). We also see that their median tends to remain stable across the different interval values, reaching approximately 670,000 seconds (i.e. less than 8 days). Even though longer interval values are intended to be better at detecting long-term training attacks, our algorithm manages to finds attack schemes unaffected by the interval values. The first and third quartiles, as well as the spread of outliers, appear to be sensitive to the interval value, but we cannot conclude about any statistical correlation. The random seeding of the genetic algorithm might be the cause.

Figure 6 shows the number of the generations where the best individuals were bred. We observe that it tends to decrease as the interval value increases. This can be explained that the fact that larger interval values naturally gives less window to be fooled but are, at the same time, more likely to detect aggressive attacks. Therefore, once a successful attack is discovered, the likelihood of finding a faster one in the upcoming generations is lower for longer interval values. For example, when the interval value is set to 19,200 or 38,400, the best attack is produced at the first 15 generations in 20 and 26 of the runs, respectively. Overall, for all interval values, the generation number of the fastest attack ranges from 1 to 101, which illustrate a relative convergence speed of our algorithm.

As a sanity check, we compared our results with a random baseline that generate the same total number of individuals (respectively as each run our our algorithm) in a single generation. Our results revealed that, with the same time budget, random search generates attack schemes that are 2-10 times slower. This tends to show that the benefit of using metaheuristics lies in finding faster attacks.

> Our testing approach can systematically find an attack scheme that deceives single-instance IDS and thereby confirms and generalizes the results of Muller et al. [11]. The median attack time is not impacted by the interval value, while the convergence of our algorithm tends to be faster for higher interval values.

5.3 RQ1.2: Searching for Training Attacks on Multiple-Instance IDS

Our second set of experiments aims to determine whether the state-of-the-art countermeasure, which recommends multiplying D-Stream instances, is resilient to our testing algorithm. To answer this question, we consider firstly all pairs of the 7 interval values used in Section 5.2. For each pair, we create an IDS composed of two parallel D-Stream instances using the interval values of the pair. Then, we test the obtained IDS against our testing algorithm and check whether a successful attack scheme was generated.

Once again, all the IDS were deceived. Detailed results are presented in Table 1 where we report, for each pair and over 50 runs, the median of the fastest attack times and the median generation numbers of those fastest attacks. With respect to the single-instance

Table 1: Median of the best attack time in seconds (left numbers) and of the number of generations at which the best attacks are found (right numbers) across 50 runs for each pair of interval values. Diagonal values denote the median for the corresponding single-instance interval value. Only upper triangle is shown since results are symmetric.

Interval values (s.)	600	1200	2400	4800	9600	19200	38400
600	(663,570 ; 31)	(662,460 ; 27)	(662,460 ; 29)	(675,120 ; 22)	(664,860 ; 26)	(674,190 ; 18)	(676,800 ; 21)
1200		(664,260 ; 34)	(662,460 ; 24)	(669,660 ; 19)	(660,090 ; 24)	(741,660 ; 18)	(671,610 ; 19)
2400			(667,260 ; 23)	(672,060 ; 20)	(664,860 ; 22)	(670,980 ; 20)	(740,430 ; 16)
4800				(664,800 ; 24)	(664,860 ;28)	(664,860 ; 17)	(670,020 ; 15)
9600					(669,330 ; 22)	(676,260 ; 16)	(676,290 ; 15)
19200						(684,000 ; 19)	(672,960 ; 18)
38400							(664,800 ; 14)

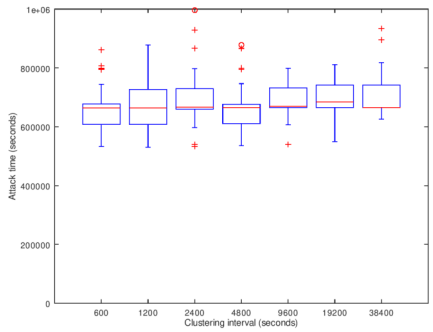

Figure 5: Time required for the best attack to deceive the system, across 50 runs and for an increasing interval value.

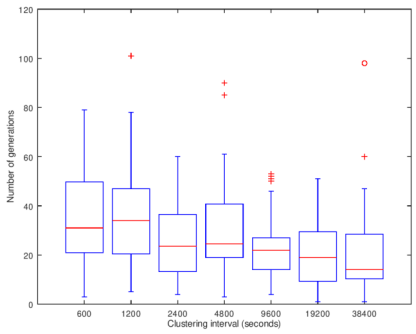

Figure 6: Number of generations required to find the best attack, across 50 runs and for an increasing interval value.

IDS, the median attack time looks unaffected by the additional instance of D-Stream. As for generation numbers, it seems dependent only on the largest interval value. This supports the generality of our testing approach and, conversely, challenges the appropriateness of the countermeasure proposed by Muller et al. [11].

To confirm these claims, we carried out another 50-run experiment where we consider 7 IDS relying on 1 to 7 parallel D-Stream instances, such that the ith IDS uses the interval values $\{600, 600\times2, \ldots, 600 \times 2^{i-1}\}$. Once again, our algorithm managed to produce successful attack schemes in every case. For example, the scheme illustrated in Figure 4 is able to deceive all the 7 IDS. We measured again the fastest attack time and the number of generations of for all the 50 runs of every tuple. Figure 7 and 8 shows the evolution of those values wrt. to the size of the considered tuples.

We see that the median attack time tends to remain stable (approximately 665,000 seconds) regardless of the tuple size. This tends to show that the state-of-the-art countermeasure is ineffective at increasing attack time. The tuple of size 6 yields a small increase (it reaches 709,920 seconds) but given that the tuple of size 7 drops down to 665,160, this is likely due to the random seeding of the algorithm. The first and third quartile do increase from the tuple of size 4 onward, though, which indicates that more parallel instances may increase the time required by the fastest and the slowest attacks.

The median generation numbers of the fastest attack found in the different runs tend to lower slightly as the tuples size grows larger. Similar observations were made when comparing single-instance IDS with different interval values. This might mean that the generation number, and thus the speed at which our genetic algorithm converges, is only affected by the largest interval values and not by the number of parallel instances.

The conclusions drawn from this experiment are stunning: the state-of-the-art countermeasure has almost no effect on the effectiveness and efficiency of our attack generation algorithm.

> Our testing approach can systematically find an attack scheme that deceives multiple-instance IDS and thereby overcomes the state-of-the-art countermeasure. Having more instances does not decrease the median attack time, while the convergence speed of our algorithm is affected only by the largest interval value.

5.4 RQ2: Searching for Defences

We consider 10 generated attack schemes that successfully deceive the best IDS considered in RQ2, i.e. the one using 7 D-Stream instances with intervals 600, 1200, 2400, 4800, 9600, 19200 and 38400 seconds. Then, for each scheme, we attempt to produce a defence strategy (i.e. a number of parallel D-Stream instances with different interval values) able to detect the executed attack. We record (a) the number of D-Stream instances used, (b) the detection times, and (c) the number of generations required to get the best strategy. To do so, we parameterized our algorithm as follows: the meta-parameters are set to $\gamma = 10, \delta = 1, \nu = 10$; the maximum number of parallel instances is set to 10; the maximum interval value is 24 hours.

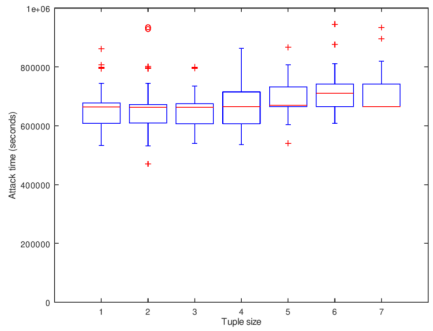

Figure 7: Time required for the best attack to deceive the system, across 50 runs and for an increasing number of D-Stream instances.

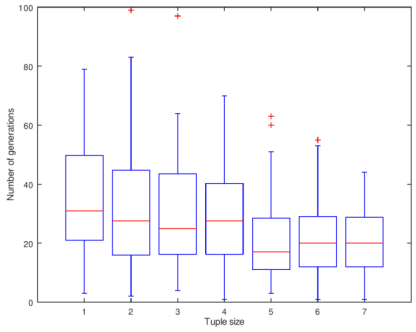

Figure 8: Number of generations required to find the best attack, across 50 runs and for an increasing number of D-Stream instances.

For every attack scheme, our defence generation algorithm managed to find a successful defence at each run. Table 2 shows, for each attack, the characteristics of the best generated defences (wrt. the fitness function as defined in Section 3.2): its clustering interval values, its size (in number of parallel instances), the time needed by the defence to detect the attack, and the number of the generation at which the best defence appeared. For conciseness, the clustering interval values are shown in minutes rather than in seconds. The corresponding attacks are not shown due to the large size of their chromosome (200 genes on average). In 4 out of the 10 cases, it returned a defence using a single D-Stream instance: given a specific attack, only one instance can be needed to detect the attack as long as it uses an appropriate interval value. The number of instances never exceeds 4, which indicates that our algorithm indeed attempts to converge towards the minimal size. In 8 of the cases, the detection time is less than 5 hours, which is significantly faster than the attack times we recorded in our previous experiments (see Figures 5 and 7). The other two defences require 3 and 4 days to counter their respective attack, but this remains faster than all the aforementioned attack times. Finally, the number of generations needed to produce the best defence varies for each case (due, again, to random seeding), but remains low overall.

Table 2: The best defences that detected the 10 attacks that were previously successful.

Defence	Size	Time (s.)	Generation
{830}	1	34,800	11
{814; 829; 791}	3	32,880	5
{839}	1	35,880	33
{1371; 1401}	2	264,240	20
{829}	1	34,680	35
{794; 1178}	2	364,020	1
{713; 815; 817; 1266}	4	33,000	25
{618; 810; 1257}	3	32,400	33
{827}	1	34,400	16
{703, 822}	2	33,840	2

Table 3: Seven successive defences generated by an execution of our co-evolution algorithm.

Defence #	Clustering Intervals
d_0	{10; 20; 40; 80; 160; 320; 640}
d_1	{919; 34; 808; 780}
d_2	{1026; 111; 855; 823; 775}
d_3	{853; 1214; 399}
d_4	{868; 1133; 773; 1349; 774; 854}
d_5	{849; 1274; 1159; 1208}
d_6	{836; 868; 1160; 1312; 322; 1032; 853}
d_7	{1423; 792; 815; 1398; 1309; 1333; 782; 607}

Our evolutionary defence algorithm succeeds in generating strategies countering effectively and efficiently multiple attack schemes that deceived manually-parameterized IDS. The algorithm tends to converge towards the minimal number of instances and can deliver a result after a small number of generations.

5.5 RQ3: Co-evolving Attacks and Defences

Our previous experiments show that our evolutionary testing framework generates effective attack schemes to deceive a particular (manually parameterized) IDS and that, conversely, our defence generation algorithm can produce strategies to overcomes those attacks even more efficiently. It remains to evaluate the capability of our co-evolution process at increasing the resilience of the IDS over multiple attacks. To answer this question, we implemented this process in our prototype tool and run it. We initialized it with the best defence we considered in RQ1, i.e. $d_0 = \{10, 20, 40, 80, 160, 320, 640\}$. We also set the number of iterations to $k = 7$. As a result, we obtained 7 new defence strategies, numbered from d_1 to d_7, which are shown in Table 3.

To evaluate these strategies, we applied our testing algorithm on d_0 50 times to obtain as many new attack schemes. We confronted the 10 successive defence strategies against the 50 schemes and computed the number of attacks that each strategy d_i detected. We ensured that none of the 50 attack schemes were used by our defence algorithm to generate any of the d_i strategies.

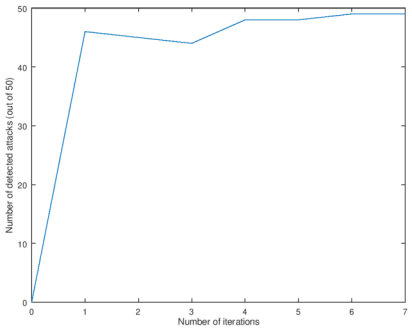

Figure 9: Number of independent attacks (out of 50) detected by the defences d_i generated by our co-evolution.

Results are shown in Figure 9. We observe that the first iteration allows us to design a defence strategy able to detect 46 of the 50 executed attacks. This number drops by one unit for each of the next two iterations. It is, however, noteworthy that d_2 and d_3 detected attacks that deceived d_1, but were themselves deceived by attack that d_1 successfully countered. Then, the number of detected attacks jumps to 48 at the 4th iteration and remains so at the 5th one. Finally, the 6th and 7th iterations yielded defences that detected 49 of the 50 attacks, the remaining one having deceived all strategies. Let us remark that these latter two strategies are also the ones relying on the highest number of D-Stream instances, which corroborates the recommendation of Muller et al [11].

> Our co-evolution process yields defence strategies able to counter most attack schemes that overcame the state-of-the-art countermeasure. The defences resulting from the highest iterations detected 49 attacks out of 50.

6 RELATED WORK

Multiple approaches based on stream clustering have been proposed to detect security anomalies. Zhong et al [20] cluster network traffic data using an online-variant of k-means, which unfortunately leads to comparatively low detection rates. Alseiari et al.[2] employ another k-means variant to cluster smart meters data and consider as anomalous every cluster smaller than a predefined threshold. They allow cluster evolution by applying a sliding time window but report that their results are sometimes unreliable. Tomlin et al. [15] use k-means and fuzzy cognitive maps to cluster security events in a power system and detect anomalies. Hendry et al.[8] propose an offline attack detection algorithm that relies on attack signatures created from recorded and clustered data. Yen et al.[19] focus on detecting malware spreading in a network, by analyzing the similarities in the online behaviour of the host.

Nevertheless, the use of machine learning to improve security expands way beyond clustering data. For example, in the field of web security, Tripp et al. [16] propose a search-based testing approach where cross-site scripting attacks are generated by learning from previous attack executions. More recently, Appelt et al. [3] present ML-Driven, an approach based on machine learning and evolutionary algorithms to test web application firewall against SQL injection attacks.

The proliferation of learning-based security mechanisms should raise awareness against the new types of attacks that exploit learning phases. Wagner and Soto [18] are the first to highlight the problem of training attacks, while Barreno et al. [4] explored the topic in more details. A related problem is the mimicry attack, which avoids detection without altering the detection system. Stevanovic et al. [14] report on real-world occurrences of such attacks on anti-DoS systems, and propose to counter them by combining a classical DoS attack monitoring system with a detection system specifically tailored for spotting mimicry attacks. In contrast to the above, other approaches rely on evolutionary algorithms to build rule-based IDS. Those search for patterns to detect certain kinds of attack, and have been applied within areas with proper attack taxonomy [12, 13].

The originality of our work lies in the combined use of both clustering and genetic algorithms to build attack-resistant live-learning IDS. Thereby, we allow those systems to benefit from live learning mechanisms while protecting them, to some extent, from their inherent vulnerabilities.

7 CONCLUSION

The rising needs for real-time adaptive security mechanisms necessitate the use of machine-learning-based anomaly detection techniques that learn the behaviour of network streams. The advantage of machine-learning-based systems is their flexibility, stability and reliability, drawn from the statistical nature of the system behaviours - one can reliably alert outliers, with a very low probability of false alarms [11]. Still, traditional IDS, such as signature scanners, are capable of detecting specific malware instances and should be applied in parallel to the machine-learning ones. Nevertheless, machine-learning-based techniques require reliable testing and configuration, as otherwise they can be fooled by training attacks.

This paper introduced a search-based approach that automatically tests and improves the attack detection capabilities of IDS. Our approach automatically generates attacks, checks their success, and incrementally learns how to fool the systems under analysis. Then, by leveraging the successful tests-attacks our approach can search for countermeasures that can successfully defend these attacks. All in all, by co-evolving both attacks and countermeasures our approach can improve the defence of the system under analysis in a reliable way (accepting normal traffic without false alarms).

Overall, this paper forms the first step towards automated solutions that can assess and improve machine-learning security systems. We dealt only with training attacks and there might be other possible techniques and opportunities to evade the IDS. However, we are confident that search-based techniques, such as ours, can be easily extended to include additional attack scenarios that can be simulated with real or synthetic data.

ACKNOWLEDGEMENTS

Mike Papadakis is supported by FNR C17/IS/11686509/CODEMATES.

REFERENCES

[1] 4SIC. [n.d.]. *4SICS geek lounge SCADA network capture.* Retrieved January 25, 2019 from http://www.netresec.com/?page=PCAP4SICS

[2] F. A. A. Alseiari and Z. Aung. 2015. Real-time anomaly-based distributed intrusion detection systems for advanced Metering Infrastructure utilizing stream data mining. In *2015 International Conference on Smart Grid and Clean Energy Technologies (ICSGCE)*. 148–153.

[3] Dennis Appelt, Cu D. Nguyen, Annibale Panichella, and Lionel C. Briand. 2018. A Machine-Learning-Driven Evolutionary Approach for Testing Web Application Firewalls. *IEEE Trans. Reliability* 67, 3 (2018), 733–757.

[4] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. 2006. Can Machine Learning Be Secure?. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (ASIACCS '06)*. ACM, 16–25. https://doi.org/10.1145/1128817.1128824

[5] R. Berthier, D. I. Urbina, A. A. Cárdenas, M. Guerrero, U. Herberg, J. G. Jetcheva, D. Mashima, J. H. Huh, and R. B. Bobba. 2014. On the practicality of detecting anomalies with encrypted traffic in AMI. In *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. 890–895.

[6] A. L. Buczak and E. Guven. 2016. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials* 18, 2 (2016), 1153–1176. https://doi.org/10.1109/COMST.2015.2494502

[7] Yixin Chen and Li Tu. 2007. Density-based Clustering for Real-time Stream Data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*. ACM, 133–142. https://doi.org/10.1145/1281192.1281210

[8] Gilbert Hendry and Shanchieh Yang. 2008. Intrusion signature creation via clustering anomalies. *Proceedings of SPIE - The International Society for Optical Engineering* 6973 (03 2008). https://doi.org/10.1117/12.775886

[9] Incapsula. 2014. *What DDoS Attacks Really Cost Business?* Retrieved 25 January 2019 from https://lp.incapsula.com/rs/incapsulainc/images/eBook-DDoSImpactSurvey.pdf

[10] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan Payne. 2015. Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices. *Comput. Surveys* 48 (09 2015), 12:1–. https://doi.org/10.1145/2808691

[11] Steve Muller, Jean Lancrenon, Carlo Harpes, Yves Le Traon, Sylvain Gombault, and Jean-Marie Bonnin. 2018. A training-resistant anomaly detection system. *Computers & Security* 76 (03 2018). https://doi.org/10.1016/j.cose.2018.02.015

[12] Mohammad Sazzadul Hoque. 2012. An Implementation of Intrusion Detection System Using Genetic Algorithm. *International Journal of Network Security & Its Applications* 4, 2 (Mar 2012), 109–120.

[13] K. G. Srinivasa, S. Chandra, S. Kajaria, and S. Mukherjee. 2011. IGIDS: Intelligent intrusion detection system using genetic algorithms. In *2011 World Congress on Information and Communication Technologies*. 852–857.

[14] Dusan Stevanovic and Natalija Vlajic. 2014. Next Generation Application-Layer DDoS Defences: Applying the Concepts of Outlier Detection in Data Streams with Concept Drift. In *Proceedings of the 2014 13th International Conference on Machine Learning and Applications (ICMLA '14)*. IEEE Computer Society, Washington, DC, USA, 456–462. https://doi.org/10.1109/ICMLA.2014.80

[15] L. Tomlin, M. R. Farnam, and S. Pan. 2016. A clustering approach to industrial network intrusion detection. In *INSuRECon '16*.

[16] Omer Tripp, Omri Weisman, and Lotem Guy. 2013. Finding Your Way in the Testing Jungle: A Learning Approach to Web Security Testing. In *ISSTA '13 (ISSTA 2013)*. ACM, New York, NY, USA, 347–357. https://doi.org/10.1145/2483760.2483776

[17] Komkrit Udommanetanakit, Thanawin Rakthanmanon, and Kitsana Waiyamai. 2007. E-Stream: Evolution-Based Technique for Stream Clustering. In *Proceedings of the 3rd International Conference on Advanced Data Mining and Applications (ADMA '07)*. Springer-Verlag, 605–615. https://doi.org/10.1007/978-3-540-73871-8_58

[18] David A. Wagner and Paolo Soto. 2002. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*. 255–264.

[19] Ting-Fang Yen and Michael K. Reiter. 2008. Traffic Aggregation for Malware Detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Diego Zamboni (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 207–227.

[20] Shi Zhong, Taghi M. Khoshgoftaar, and Shyarn V. Nath. 2005. A Clustering Approach to Wireless Network Intrusion Detection. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '05)*. IEEE Computer Society, Washington, DC, USA, 190–196.