Computer Networks 127 (2017) 200–216

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Toward a reliable anomaly-based intrusion detection in real-world environments

Eduardo K. Viegas[a], Altair O. Santin[a,*], Luiz S. Oliveira[b]

[a] Pontifical Catholic University of Parana, Brazil
[b] Federal University of Parana, Brazil

ARTICLE INFO

Article history:
Received 23 December 2016
Revised 11 July 2017
Accepted 14 August 2017
Available online 16 August 2017

Keywords:
Multi-objective feature selection
Anomaly-based classifier
Machine learning-based intrusion detection
Intrusion databases

ABSTRACT

A popular approach for detecting network intrusion attempts is to monitor the network traffic for anomalies. Extensive research effort has been invested in anomaly-based network intrusion detection using machine learning techniques; however, in general these techniques remain a research topic, rarely being used in real-world environments. In general, the approaches proposed in the literature lack representative datasets and reliable evaluation methods that consider real-world network properties during the system evaluation. In general, the approaches adopt a set of assumptions about the training data, as well as about the validation methods, rendering the created system unreliable for open-world usage. This paper presents a new method for creating intrusion databases. The objective is that the databases should be easy to update and reproduce with real and valid traffic, representative, and publicly available. Using our proposed method, we propose a new evaluation scheme specific to the machine learning intrusion detection field. Sixteen intrusion databases were created, and each of the assumptions frequently adopted in studies in the intrusion detection literature regarding network traffic behavior was validated. To make machine learning detection schemes feasible, we propose a new multi-objective feature selection method that considers real-world network properties. The results show that most of the assumptions frequently applied in studies in the literature do not hold when using a machine learning detection scheme for network-based intrusion detection. However, the proposed multi-objective feature selection method allows the system accuracy to be improved by considering real-world network properties during the model creation process.

1. Introduction

Machine learning techniques for performing anomaly-based network intrusion detection have been extensively studied over the years. Despite the extensive and promising results reported in the literature [1], pattern recognition in intrusion detection remains mostly a research topic, rarely being deployed in real-world (production) environments [2].

Typically, the main reason for using machine learning is the assumption that it can detect new attacks. This is achieved by specifying only the expected (normal, background traffic) while considering the remaining activity as an intrusion attempt. However, when using a machine learning technique, during the system training examples from all classes are normally required. In other words, an anomaly-based system using a machine learning technique must have a significant number of examples from all the considered classes, and thus requires representative training examples [3].

The lack of public and updated training datasets, as well as of specific evaluation methods that take the intrusion detection properties into account, makes it difficult to adopt anomaly-based intrusion detection in production environments. A typical machine learning evaluation scheme relies on a test dataset. It assumes that the classifier's accuracy rate obtained in the test dataset is the same as the accuracy rate obtained during its usage in production environments.

Unlike other fields, where machine learning is extensively used, the field of intrusion detection involves significantly different characteristics. Because of the highly variable and constant changes in open-world network traffic behavior [4], to create a representative training dataset is a difficult task. Thus, normally researchers incorrectly assume an immutable network traffic behavior and thus evaluate the conceived system on a single dataset, discarding the open-world network traffic behavior characteristics [5], such as

* Corresponding author.
E-mail addresses: eduardo.viegas@ppgia.pucpr.br (E.K. Viegas), santin@ppgia.pucpr.br (A.O. Santin), luiz.oliveira@ufpr.br (L.S. Oliveira).

http://dx.doi.org/10.1016/j.comnet.2017.08.013
1389-1286/© 2017 Elsevier B.V. All rights reserved.

non-site-specific, new (different) attack behaviors, highly variable content (service), and continuously occurring changes in the traffic behavior.

Several techniques have been proposed in the literature for the creation of an intrusion dataset [3,6,7]. However, the most frequently used dataset [8] remains the well-known DARPA1998 dataset [9], which is now almost 20 years-old. Most approaches used to create a public intrusion dataset attempted to statistically model the user behavior [10]. Normally, a typical and real user is monitored during a certain period and its traffic characteristics are reproduced in a statistically similar manner. Thus, a static user behavior is imposed during the monitored period. Thus, these approaches generate a site-specific traffic behavior that are difficult to reproduce.

The great majority of research studies on the subject were aimed to improve the classifier accuracy on a specific dataset [3,11,12], usually on KDD'99 [13], which was created in 1999, through the DARPA1998 database, with several limitations [14,15], making the obtained results unrealistic [1]. Moreover, little or no effort has been invested in using the obtained models in real-world environments [2].

In addition to the problems inherent in the datasets that are used, other factors exist that indicates that the reported results are not reliable. There are no guarantees that the accuracy obtained during the model evaluation is also obtained in real-world environments; the situation is aggravated when the dataset used for creating the model is obsolete.

Independently of the used dataset, the evaluation and model creation method must consider the network intrusion environment characteristics. An open-world network traffic changes continuously in terms of service, traffic content, attack, and scenario arrangement. New attacks are discovered every day, and their behavior, and even that of known attacks, may change as a result of evasion techniques. Thus, it cannot be assumed that the network traffic behavior evidenced in a training dataset is to any extent similar to that in a production environment over time.

In contrast to other detection fields, intrusion detection must detect intrusion attempts with high accuracy rates because of the high cost of errors [1]. Therefore, during the creation process of the intrusion model, researchers normally perform feature selection to improve its accuracy [3,13]. However, in general, because of the restrictions in the dataset that they used, the authors of those studies did not consider the network characteristics during the feature selection process. The main contributions of our paper are as follows.

- We propose a tool-based method that produces real and valid network traffic in a controlled and reproducible environment for creating intrusion databases. Our proposed method allows the created database and dataset to be shared, as the collected data contain no sensitive data. The variability problem inherent in creating databases in a controlled environment [2] is reduced by creating a complex but easy-to-reproduce client behavior. The attack traffic generation problem is solved by using well-known standardized tools. Our method thus allows some of the main problems raised by the authors of [6] [7] regarding database creation for intrusion detection to be solved.
- We examine in depth and propose a new and rich evaluation method specific for the machine learning anomaly-based intrusion detection field, using our tool-based method to create intrusion datasets. Our evaluation method provides ratings of the performance of a system for different normal and attack behaviors and scenario arrangements, used in production environments.
- We propose and evaluate a new multi-objective feature selection method that considers the network properties in intrusion detection through using machine learning algorithms. Our proposed multi-objective feature selection method allows a system administrator to improve the system accuracy according to its needs considering the different normal, attack, and scenario behaviors.

The remainder of this paper is organized as follows. Section 2 presents the background. Section 3 provides a brief overview of the related work on intrusion dataset creation and evaluation methods. Section 4 discusses the details of our proposed method for intrusion dataset creation, our proposed evaluation method, and our multi-objective feature selection method. In Section 5 an evaluation of our proposed method is presented. Finally, Section 6 concludes this paper.

2. Background

An intrusion detection system (IDS) is software that is aimed to identify and classify malicious activities in an environment [3]. A typical IDS is composed of four modules. (i) *Event gatherer*: this module is responsible for collecting events from the environment for further monitoring, e.g., reading network packets from a network interface card. (ii) *Preprocessing*: this module performs the processing needed before the detection engine is run on the collected events, e.g., extraction of a set of features. (iii) *Detection*: this module is based on the preprocessed event that the detection engine uses to decide whether an event is normal or an intrusion attempt. (iv) *Alert*: finally, if an event is considered as an intrusion attempt, this module generates an alert.

Several approaches for classifying intrusion attempts have appeared in the literature. Currently, the most frequently used taxonomy, which is used in DARPA's intrusion datasets, was defined by Kendall [9]. According to Kendall, intrusions can be divided into four classes as follows. (i) *Probing* attacks that gather information about a target; (ii) *denial of service* (DoS) attacks, that is, any attempt to prevent legitimate users from accessing a service or system; (iii) *remote to local* (R2L) attacks that attempt to gain access to a legitimate user's account on a system; and (iv) *user to root* (U2R) attacks, which occur when the attacker already has achieved access to a legitimate user's account and then attempts to obtain administrator privileges.

An IDS can be either network-based (an NIDS) or host-based (an HIDS); the IDS type determines which attack class it is able to detect. An NIDS performs the detection at the network level, detecting attacks such as a probing (e.g., port scan) and a network-based DoS attacks (e.g., synflood). Thus, an NIDS accesses the data only at the network level, e.g., network packets or network flows, to perform the event classification. In contrast, an HIDS detects application-based attacks, such as R2L (e.g., buffer overflow) and U2R (e.g., privilege escalation). An HIDS requires knowledge about the applications running on the protected host, thus requiring access to logs and other data about the system in order to perform its detection.

In general, two approaches can be used during the detection stage: signature and anomaly. The signature-based approach consists of searching well-known attack patterns in the received events. Each detected event is matched against the signature database; if a known attack pattern is found, the event is classified as an intrusion attempt. When a new vulnerability is discovered and reported, a related signature is created. The main drawback of the signature-based approach is its inability to detect unknown attacks.

In contrast, the anomaly-based approach is aimed to detect new (unknown) attacks by modeling the activities that are considered normal within a system and identifying possible attacks from behaviors that deviate from the known normal behavior pattern [2].

The event behavior is defined by a set of features extracted during the preprocessing stage. In studies in the literature, anomaly-based intrusion detection was in general treated as a pattern–recognition problem [16] through machine learning techniques.

Machine learning is aimed to assign a class (e.g., normal or attack/intrusion) to an event. In the process, first events captured from the environment are stored in a database. From each event in the database, a set of features is extracted and stored in a dataset. A machine learning algorithm is then used to infer a pattern from the dataset and to create a model that represents such behavior. However, events that present a behavior similar to that of other classes may be wrongly classified, e.g., an attack that is similar to a legitimate request.

An intrusion model is created using a training dataset; for the estimation of the classifier accuracy rate, a validation dataset is used. The validation dataset is utilized for making possible improvements to the intrusion model. The final version of the intrusion model is evaluated through a test dataset. For the process to be reliable, each used dataset must contain different events. During the tests, the false-positive (FP) and false-negative (FN) rates are estimated. An FP occurs when a legitimate activity is classified as an intrusion, whereas an FN occurs when an intrusion is classified as a legitimate activity.

The set of features used to obtain the intrusion model must be discriminant because, on the basis only of the extracted set of features, it must be possible to correctly classify events from different classes. The feature selection process is aimed to identify the most relevant features in a feature set, allowing the classifier accuracy to be improved and the computational load during the event classification to be reduced. During the feature selection process, two approaches may be applied: (i) filter, in which the selected features are independent of the used classifier, or (ii) wrapper, which is classifier-dependent. The wrapper approach incurs the computational overhead of evaluating the candidate feature subsets by executing the machine learning algorithm on the dataset, using each subset under consideration.

Because a complete search over all possible subsets of a feature set (2^N, where N is the number of features) may not be computationally feasible, several authors have explored heuristics for feature subset selection (e.g., the intrusion detection accuracy). Genetic algorithms (GAs) are an interesting approach, because they do not assume restrictive monotonicity and can use multi-objective optimization.

Solutions to a multi-objective optimization problem can be expressed mathematically in terms of non-dominated points; a solution is dominant over another only if its performance is superior for all the considered objectives. A solution is said to be Pareto-optimal if it cannot be dominated by any other solution available in the search space. A popular multi-objective GA that has been successfully applied to multi-objective feature selection [17] is the non-dominated sorting genetic algorithm (NSGA-II) [18]. NSGA-II uses a ranking selection method to emphasize good points and a niche method to maintain the stable subpopulations of the good points. It varies from simple GAs only in the operation of the selection operator.

In brief, a detection system's reliability relies on an appropriately obtained training dataset. The expected properties for an intrusion dataset are as follows [7]. (i) *Realism*: the dataset should contain network traffic that can be observed in production environments; (ii) *validity*: the dataset should contain well-formed packets, with a complete client-server interaction; (iii) *prior labeling*: in the dataset, each event must be correctly labeled (as belonging to a class, e.g., normal or attack), to allow correct classifier training; (iv) *diverse/high variability*: the dataset should present a diversity of services, client behaviors, and attacks; (v) *correct implementation*: in the dataset, the attacks must follow a well-known or "de facto" standard; (vi) *ease of updating*: the dataset should be able to incorporate new services and attacks that are discovered every day; (vii) *reproducibility*: the dataset should allow experts to perform a comparison between datasets; and, finally, (viii) *without sensitive data*: the dataset should not reveal sensitive data to allow the free dataset to be shared among researchers.

Two approaches may be used for obtaining the datasets for an NIDS: in the first, the production environment is recorded and in the second a controlled environment is created [19]. Production environment monitoring allows traffic that is real and similar to this environment to be obtained. However, because of privacy concerns it is not feasible to share the dataset. Moreover, the creation of a database in a controlled environment using tools allows it to be shared freely, but the approach suffers from traffic invariability problems [1]. The anomaly-based IDS literature lacks a ground-truth dataset.

3. Related work

Many research studies have been conducted since the anomaly-based detection paradigm was introduced by Denning [20]. However, despite the extensive amount of work, few applications of any anomaly-based intrusion detection systems in production environments have been reported. In recent years, some researchers have begun to question the applicability of the results provided in the literature. Gates and Taylor [1] argued that only a few anomaly-based IDS have been widely used. They considered mainly the assumptions originating in Denning's work. According to them, the lack of appropriately obtained training data and testing methodologies that consider the network properties, such as continuous changes in content, volume, and attacks, is the main reason why the anomaly detection approach is unsuccessful.

Sommer and Paxson [2] conducted an extensive review of intrusion detection. They argued that the field is significantly different from other fields in which the machine learning approach has been successfully applied. They claimed that machine learning is more effective for finding similarities than detecting outliers, for instance. The high cost of errors inhibits its use in production environments. The lack of available public and updated data hinders an appropriate system evaluation and comparison [21,22]. Sommer and Paxson [2] and Paxson and Floyd [4] reported that real-world environments present a significantly different behavior from the data the systems are normally trained.

The reliability of an anomaly-based detection system relies on an appropriately created training dataset. Normally, certain strong assumptions about the training data need to be adopted. Canali et al. [23] created their dataset by collecting several Website contents from the Internet; they labeled each datum by using state-of-the-art tools and manually inspecting the data to ensure that the Website contents were correctly labeled. The authors assumed that most of the frequently visited Websites worldwide are benign and that the distribution of feature values is different for each class of Website. The strongest assumption is that the dataset used to train the models presents the same feature distribution as real-world environments.

Moreover, when a dataset is obtained in a controlled environment, the authors normally statistically reproduce the user behavior. Shiravi et al. [7] created user profiles on the basis of the user behavior for each application during an observed time interval. Kendall [9] created a dataset by statistically reproducing the user behavior in an air force environment. In general, these approaches lack upgradeability, wrongly assuming that network traffic is immutable and considering that the user behavior can be modeled [1,2]. Thus, the approach we propose in this paper is aimed to provide a publicly available intrusion database through

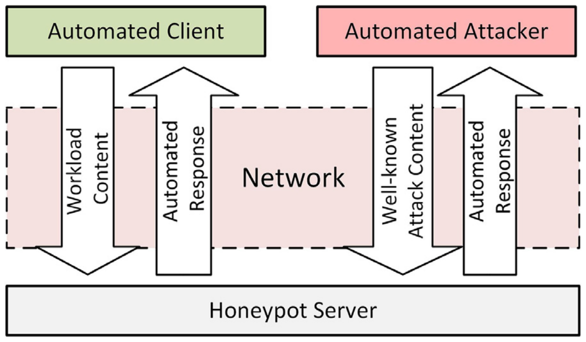


Fig. 1. Proposed database creation method.

the use of well-known tools in a controlled environment, thus providing the properties expected from an intrusion database.

Multi-objective feature selection has been addressed in several fields, such as Quality of Service (QoS) constraints in multimedia content distribution [41], handwritten digit recognition [42], among others [43]; however, for intrusion detection purposes its use remains rare. De la Hoz et al. [40] used a similarity measurement of the predicted labels and the ground truth dataset for each considered class as a different objective to be maximized. The authors were able to improve the accuracy while decreasing the number of features; however, they relied in the outdated KDD99 dataset.

Viegas et al. [3] proposed a feature selection approach for finding the best tradeoff between the intrusion detection accuracy and the system energy consumption. The authors' approach was able to provide up to a 93% reduction in energy consumption while incurring only a 0.9% accuracy loss. To the best of our knowledge, this study is the first to address the production environment properties as a multi-objective problem in the intrusion detection field.

4. Proposed methods

In this paper, we propose a new intrusion database creation method that is aimed to present the properties expected from an intrusion dataset. Moreover, through this proposed method, a new and extensive evaluation method specific to the machine learning field is proposed to validate the common assumptions in the literature regarding NIDS. Finally, a new multi-objective feature selection method is proposed, which considers the expected anomaly-based properties during the model creation process.

4.1. Database creation method

The proposed intrusion database creation method is aimed to ensure that a database contains the properties expected from an IDS testing intrusion database. To achieve this, the proposed method creates intrusion databases in a controlled environment and reproduces a user's behavior through well-known tools. The method considers two different users: the legitimate client and the attacker. The traffic is generated considering the client-server model. For generating the server-side network traffic, the honeypot technique is used, whereas the client traffic is generated through real-world workload tools. Thus, real and valid traffic is generated for the client-server communication. The attacks are generated using known, standardized, and widely used tools frequently implemented for system auditing. The traffic creation method is shown in Fig. 1 and further described in the following sections.

4.1.1. Normal (background) traffic creation method

It is recommended that the traffic included in an intrusion database used for anomaly-based detection systems be real and valid. Thus, the method used for its design must ensure that the client-server interaction occurs correctly, thereby guaranteeing that the client behavior evidenced in the database is similar to that observed in the real world.

The normal (legitimate) traffic must be generated according to two perspectives: the client and the server. The client is responsible for requesting the services available on the server, whereas the server is responsible for providing the appropriate response to each request in terms of content and service behavior. It is expected that the provided services, as well as their request contents, are highly variable.

Independent of the considered application, each user network traffic behavior, in general, is random and does not follow a statistical distribution when compared to that of a different application user [24,25]; for instance, the behaviors of two different users browsing a Web application are not similar. Thus, the proposed method generates the normal (background) traffic by providing a set of services, where each service has a set of contents that may be requested (Fig. 1). Each client performs a real and valid request through a real-world workload tool; thus, real, and valid requests are generated for the client side. Each client sends a request for a previously defined service and a specific service content. After the client-server communication, the client waits a variable time and transmits a new request for another service. Thus, the client behavior is modeled according to the observed application usage, e.g., a client browsing a Web page for a certain duration.

In turn, the server must be able to interpret the received request and generate the appropriate reply. The use of tools specific to the service being provided makes it difficult to update the intrusion database, whereas the use of a technique that mimics the server responses allows the database to be updated easily. Thus, the honeypot technique was considered, which allows the responses of a vulnerable server to be mimicked, with automated and valid responses to be generated.

A set of predefined services are provided. Thus, every request, regardless of the requested service, is correctly interpreted and receives a legitimate reply. Under this assumption, the proposed method generates real, valid, and easy-to-update background traffic (the expected properties are described in Section 2). The complete background traffic generation process is shown on the left side of Fig. 1 (Automated Client).

4.1.2. Attack traffic creation method

The lack of an appropriate implementation guarantee is the main problem that has been reported to occur during attack generation. In general, researchers (see, e.g. [10]) implement a known attack according to their discretion, which makes an attack difficult to reproduce as there is no guarantee that the implementation follows a well-known defined standard.

Immediately after a new attack becomes known and is reported, entities specify it. Initiatives such as the Common Vulnerabilities and Exposures (CVE) include the details of new vulnerabilities and the affected services. Implementations that are, for example, CVE-compatible, guarantee that an attack will behave as expected (reported). Thus, tools that follow well-known standards are auditable and can be assessed.

Our approach, unlike those where the authors implemented their own version of the attacks, is based on the use of well-known and de facto standardized tools to generate the attacks. This approach ensures that the implementation of all the attacks included in the database is dependable when they become public, as the approach follows a well-known standard. The complete attack traffic generation process is shown on the right side of Fig. 1 (Automated Attacker).

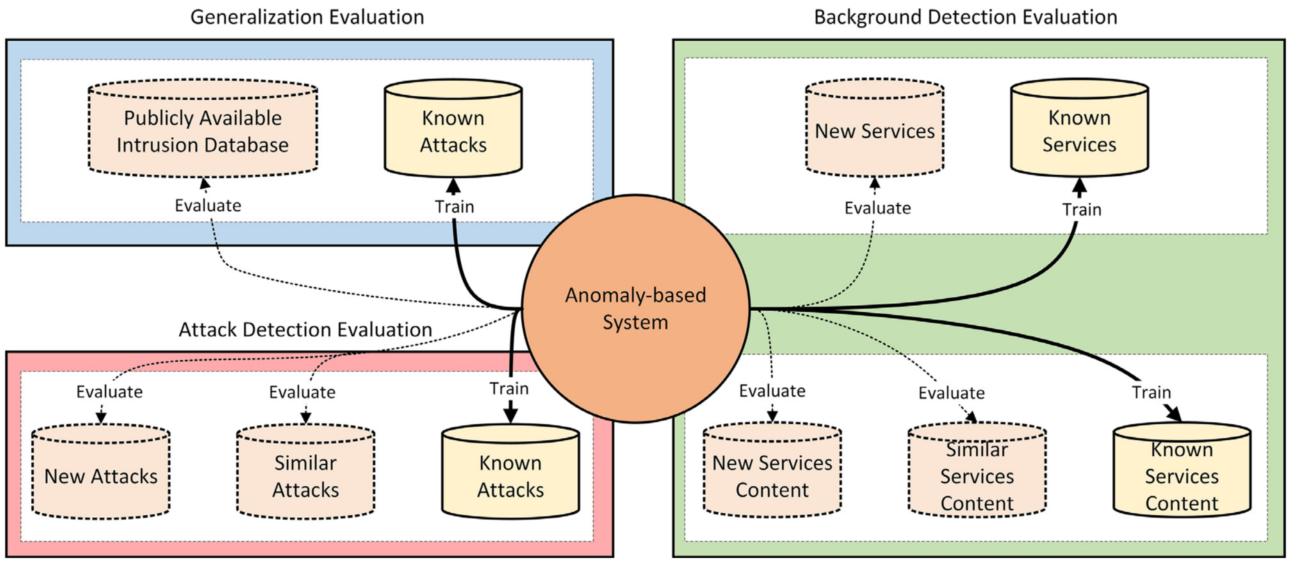


Fig. 2. Proposed evaluation method.

4.2. Anomaly-based intrusion detection evaluation method

Three steps are normally involved in a typical machine learning evaluation method. Initially, the classifier model is created using a trained dataset. Then, a validation dataset is used to improve the created model. Finally, the model is evaluated by means of a test dataset. Because of the lack of publicly available data in the NIDS field, experts normally divide a single dataset into three parts. Thus, a typical evaluation method assumes that the used datasets resemble the production environment.

However, this assumption does not hold in networked environments. The creation of an intrusion database that presents all the possible behaviors in a production environment is not a feasible task [1]. Even if it were possible to create a perfect database, it would still not be effective because it would consider that network traffic is immutable [2]. Thus, the proposed evaluation method is aimed to validate the expected properties for any machine learning algorithm when applied to the anomaly-based intrusion detection area.

It is a well-known problem that the intrusion detection research community lacks public datasets for appropriately evaluating developed systems [26–29]. We propose a new evaluation method specific to the intrusion detection field that uses our proposed database creation method (described in Section 4.1). The purpose is to validate the common assumptions reported in [1,2]. The overall process is shown in Fig. 2 and further described in the following sections.

4.2.1. Attack detection rate

The most important assumption about an anomaly-based intrusion detection system is that it is capable of detecting new attacks. The premise is that an attack, whether new or known, shows behavior that is significantly different from that of a typical system's usage and, thus, can be identified by detecting outliers [30]. However, incoming events are classified, in general, on the basis of their similarity to the known and prior-labeled events in the training dataset, according to a similarity metric.

Thus, only new attacks that behave similarly to already known attacks can be correctly classified. By definition, it is not possible to train a machine learning detection technique using unknown attacks. However, it is possible to measure an intrusion detection system's capability by controlling the events included in the test datasets. A system can be trained with a limited type of attack and tested with similar or completely different attacks. The definition of the similarity of events is context-dependent and must be determined according to expert knowledge (described in Section 5). The process is shown in Fig. 3.

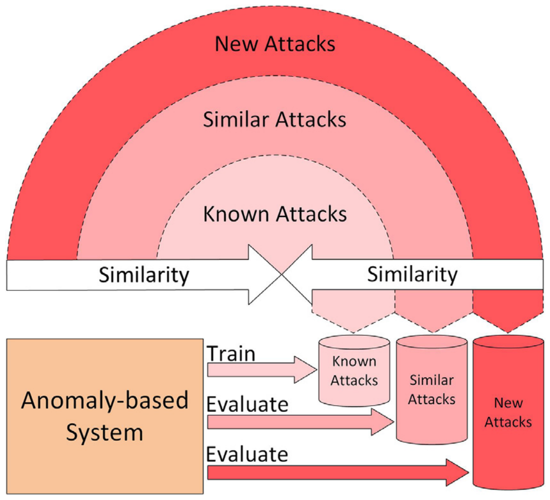


Fig. 3. Proposed attack detection evaluation method.

Initially, an anomaly-based intrusion detection system is trained with a limited set of attacks that are similar and present an expected behavior during the system usage in production environments (Fig. 3, Known Attacks). Then, the created detection model is evaluated using databases containing similar and new attacks (Fig. 3, Similar and New Attacks). Similar attacks in this context are attacks with a high similarity to the attacks on which the system was trained, whereas new attacks are attacks that are significantly different from known attacks.

The similar attack detection rate is defined as the system's capacity to detect events with behaviors similar to those of known attacks. This property is desirable in intrusion detection systems because of the highly variable nature of networked environments. Thus, a system must be able to cope with similar attacks, as a single database cannot contain all possible attacker behaviors. Similar attacks may present a different pattern and can evade signature-based systems, where detection is performed by matching against well-known attack patterns. However, similar attacks may present the same or a similar behavior and should be detected by an anomaly-based detection scheme if the used features are adequately discriminant.

The new attack detection rate defines the system capacity to detect significantly different types of attacks; i.e., attacks that present a behavior that is completely different from any known behavior on which the system was trained. This type of incidence occurs in production environments, as it is not possible to train a system with all known or new attacks. The used detection scheme must be able to relate the new attack behavior to the known attacks and correctly detect it, which is the premise when using any anomaly-based detection approach.

During database creation for machine learning detection schemes, the background traffic must also be generated. The incidence of an attack affects the response of a service to legitimate requests; e.g., one attack type can make a service unavailable, whereas another can make a service reply only to a specific set of requests. Thus, the occurrence of an attack can affect also the background traffic detection rate.

Note that databases must use the same background traffic creation approach. Thus, this approach uses the background traffic as the baseline, allowing it to identify the behavior of the same set of services under each type of attack and the performance of the detection engine while detecting such changes.

4.2.2. *Background detection rate*

The background traffic is composed of two entities: the client and the server. The client generates requests according to the contents and services provided by the server. Thus, the background traffic may vary in content and the service requested.

Because of the lack of publicly available data, researchers have assumed that network traffic is immutable [31]. Anomaly-based systems are normally tested using a single database, divided into three parts having the same background traffic behavior.

It is known that network traffic continuously changes [1,2]. Thus, it cannot be assumed that a behavior evidenced in a certain period during the intrusion database creation will remain immutable over time. The detection scheme must be able to track the background traffic behavior changes while still performing its detection at a reasonable rate, when a classifier model update is not possible.

When evaluating an intrusion detection method, one must consider the database limitations. It is not possible to create every service and content that will be evidenced in production environments. Thus, it is not feasible to create every possible background traffic behavior. However, a detection scheme must be able to detect different background traffic contents and different services, with their new type of content. To present such properties, the evaluation databases must be modified to allow them to be tested in an anomaly-based system.

Similar to the attack detection rate method (Section 4.2.1), the background detection rate method consists of restricting the used databases to two perspectives: the service content (Fig. 4) and the services (Fig. 5).

The service content detection rate is established by limiting the client requests, which are divided into three groups: known, similar, and new content (Fig. 4). However, the service detection rate is established by restricting the number of services in the training database dataset and evaluating the classifier with a new set of services not used during the training stage (Fig. 5).

In the evaluation of the background detection rate, the attacks are used as the baseline. The same set of attacks is generated in each scenario, allowing the attack occurrence to be evidenced in each used service and its content.

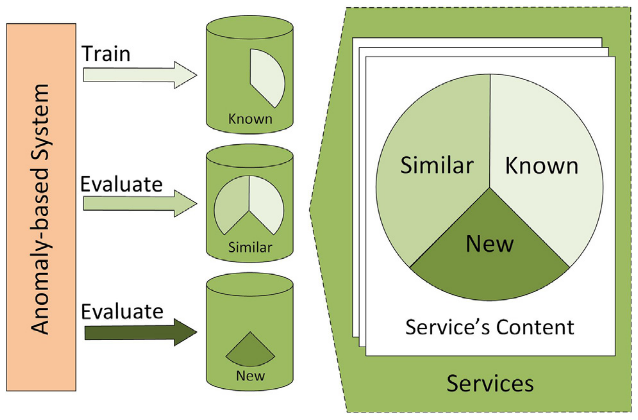


Fig. 4. Proposed background service content detection evaluation method.

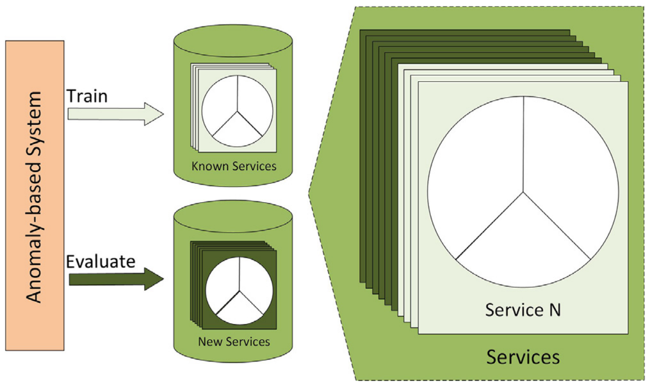


Fig. 5. Proposed background service detection evaluation method.

4.2.3. *Generalization evaluation*

Several methods have been proposed in the literature for creating intrusion databases; however, despite extensive efforts, they are all exposed to the problems inherent in the method used for their creation [32]. Thus, to evaluate the database used during the system design, as well as the method used for event detection, we need an evaluation method that uses a publicly available intrusion database. Thereby, the system evaluation using a publicly available database provides a baseline comparison reference and the generalization rate.

The generalization is a desirable property for any machine learning technique. The set of extracted features must allow the classifier to generalize the problem appropriately by distinguishing the classes, regardless of the current environment in which it is operating. Thus, the classifier model built from the set of extracted features may be used in other environments that aim to detect the same type of events.

In this way, evaluation that uses a publicly available database ensures that the conceived anomaly-based detection scheme can operate independently of the environment in which it was conceived.

4.2.4. *Evaluation method summary*

The anomaly-based intrusion detection field faces challenges that are significantly different from those of other areas where machine learning has been successfully applied [2]. The proposed evaluation scheme is aimed to test the expected properties of an anomaly-based machine learning intrusion detection scheme. The following properties can be provided by the proposed evaluation method (Fig. 2):

- Detection rate for known, similar, and new attacks (Section 4.2.1);
- Detection rate for known and new services (Section 4.2.2);
- Detection rate for known, similar, and new services' content (Section 4.2.2);
- Detection rate while operating in a different environment (Section 4.2.3).

4.3. Multi-objective feature selection for intrusion detection

To improve the detection rates mentioned above (Section 4.2.4), we propose a multi-objective feature selection method specific to the intrusion detection field. This method considers that, during the system development, the system designer takes into account the following detection properties of the detection system: *attack* ($attack_{rate}$), *normal* ($normal_{rate}$), and/or *generalization* ($generalization_{rate}$).

As described in Section 4.2.1, an intrusion detection system may face three distinct attack behaviors in production environments: *known*, *similar*, and *new*. During the classifier training, the detection algorithm learns only the *known* behavior. However, in production environments, the probability of each attack behavior occurring is unknown. For instance, an IDS that was trained with a network-based DoS attack behavior (*known*) may also face application-level DoS attacks (*similar*) having an unknown occurrence probability. However, in the production environment, the system administrator expects that an intrusion detection engine is able to detect attacks according to the accuracy rate obtained during the classifier testing, regardless of the current attack type the system is facing. Thus, the *attack* detection rate in a production environment can be calculated as

$$attack_{rate} = average\begin{pmatrix} attack_{rate}^{known}, \\ attack_{rate}^{similar}, \\ attack_{rate}^{new} \end{pmatrix} \tag{1}$$

where $attack_{rate}^{known}$ denotes the system detection rate for known attacks, $attack_{rate}^{similar}$ denotes the detection rate for similar attacks, and $attack_{rate}^{new}$ denotes the detection rate for new attacks (Fig. 2, Attack Detection Evaluation). Thus, the system attack detection rate ($attack_{rate}$) is represented by the average detection rate of *known*, *similar*, and *new* attacks in production environments.

The same property is expected in a normal (background) traffic perspective. The requested service, either *known* or *new*, must be correctly detected, as well as its content: *known*, *similar*, and *new*. Thus, the *normal* detection rate is established by

$$normal_{rate}^{service} = average\begin{pmatrix} normal_{rate}^{known\ service}, \\ normal_{rate}^{new\ service} \end{pmatrix} \tag{2}$$

$$normal_{rate}^{content} = average\begin{pmatrix} normal_{rate}^{known\ content}, \\ normal_{rate}^{similar\ content}, \\ normal_{rate}^{new\ content} \end{pmatrix} \tag{3}$$

$$normal_{rate} = average\begin{pmatrix} normal_{rate}^{service}, \\ normal_{rate}^{content} \end{pmatrix} \tag{4}$$

where $normal_{rate}^{known\ service}$ denotes the system detection rate for known services, $normal_{rate}^{new\ service}$ denotes the detection rate for new services, and $normal_{rate}^{known\ content}$, $normal_{rate}^{similar\ content}$, and $normal_{rate}^{new\ content}$ refer to the detection rate of known, similar, and new services' content, respectively. The system's normal detection rate ($attack_{rate}$) is represented by the average detection rate of $normal_{rate}^{service}$ and $normal_{rate}^{content}$.

Finally, the generalization capacity of a system is directly established by the system detection rate in another environment (Fig. 2, Generalization Evaluation). Thus, the generalization rate ($generalization_{rate}$) is established by

$$generalization_{rate} = generalization_{rate}^{public\ database} \tag{5}$$

It is important to note that *attack* ($attack_{rate}$), *normal* ($normal_{rate}$), and *generalization* ($generalization_{rate}$) are conflicting properties (objectives). For instance, an increase in $generalization_{rate}$ may decrease the intrusion detection rate for normal and attack events because of the increase in the generalization capacity (commonly referred to as the receiver operating characteristic curves for two class decision systems) [33].

Thus, the operating points must be established according to the system designer's needs. For example, the generalization property may be desired in systems that will be used in several different environments (commercial products for instance); however, in proprietary systems, this property may not be desired. The operating points and evaluation of objectives are further described in Section 5.4.

5. Evaluation

In this section, we present the evaluation of our proposed methods for intrusion database creation, anomaly-based system evaluation, and multi-objective feature selection. Specifically, we first discuss the scenarios for each database used in our work that are used in the evaluation method described in Section 4.2. Then, using our proposed evaluation method, we present the evaluation of a set of methods for performing anomaly-based intrusion detection to confirm whether the assumptions in the intrusion detection literature do hold. Finally, the evaluation of the multi-objective feature selection method for intrusion detection systems is presented (Section 4.3).

5.1. Intrusion database creation

The following sections discuss the details of the application of the intrusion database method described in Section 4.2. An extensive description of the background and attack traffic generation is provided. Then, the testbed network infrastructure is discussed in detail.

5.1.1. Background traffic generation

The most desirable property of an intrusion database is that the background traffic is as realistic as possible. The normal traffic must be highly variable, real, and valid. Background traffic generation is a complex and difficult task, mainly because of the complexity involved in modeling user behavior [34], which is, in general, random and application-dependent. The network traffic generated is dependent on the user demand for an application and is specific to the environment being reproduced.

Taking these factors into account, we treated each client as an entity with a pseudo-random behavior that does not follow any statistical distribution pattern, as reported in [4]. Each of our clients showed a unique behavior when requesting a service. Each client might request one or more services.

To achieve this property, we established a set of predetermined services on the basis of the frequently used services discussed in [35]. The following services were considered to be generated on the testbed environment: HTTP, SNMP, SMTP, NTP, and SSH. Every name resolution (DNS) was also generated as a consequence of using the listed protocols.

To create the honeypot server (Fig. 1), which executes the server-side applications, in the proposed method we used the Honeyd (www.honeyd.org) tool. To develop the client-side application for use with the servers, we used a workload tool as the service request tool. It is important to emphasize that the only

Table 1
Services used for the background traffic generation.

Service	Description (Client behavior varying from 0 to 4 s interval)
HTTP	The 1000 most visited Websites worldwide were downloaded using www.alexa.com/topsites and hosted on the honeypot server; each HTTP client requests a pseudo-random Website from this set of contents.
SMTP	Each SMTP client sends a mail with 50–400 bytes in the subject line and 100–4000 bytes in the body.
SSH	Each SSH client logs in to the honeypot host and executes a random command from a list of 100 possible commands.
SNMP	Each SNMP client walks through a predefined management information base (MIB) from a predefined list of possible MIBS.
NTP	The client performs time synchronization through the NTP protocol.
DNS	Every name resolution was also made to the honeypot server

Table 2
Tools used for attack network traffic generation.

Category	Attack type	Tool used	Description
DoS	SYN flood	Hping3	Sends several requests to open TCP connections, varying the attack send frequency and the duration time
	UDP flood	Hping3	Sends several UDP datagrams to an open DNS port, varying the attack send frequency, the duration time, and the size of each datagram
	ICMP flood	Hping3	Sends several ICMP messages to the target, varying the attack send frequency, the duration time, and the size of each datagram
	TCP keepalive	Slowloris	Initiates several HTTP connections and keeps them open for a period, varying the number of connections to be opened
	SMTP flood	Postal	Sends several emails to an SMTP server, varying the duration time, body size, subject size, and frequency
	HTTP flood	LOIC	Sends several HTTP-get requests to a specific URL, varying the duration time and frequency
Probing	UDP scan	Nmap	Searches for open UDP ports, varying the attack send frequency and the duration time
	SYN scan	Nmap	Searches for open TCP ports by sending TCP packets with the SYN flag set while varying the attack send frequency and the duration time
	NULL scan	Nmap	Searches for open TCP ports by sending TCP packets without any flags set while varying the attack send frequency and the duration time
	TCP connect	Nmap	Searches for open TCP ports by completing the three-way handshake while varying the attack send frequency and the duration time
	FIN scan	Nmap	Searches for open TCP ports by sending TCP packets with the FIN flag set while varying the attack send frequency and the duration time
	XMAS scan	Nmap	Searches for open TCP ports by sending TCP packets with the FIN, PSH, and URG flags set while varying the attack send frequency and the duration time
	ACK scan	Nmap	Searches for open TCP ports by sending TCP packets with the ACK flag set while varying the attack send frequency and the duration time
	OS Fingerprint	Nmap	Identifies the OS from the target (https://nmap.org/book/osdetect.html) while varying the attack send frequency and the duration time
	Service Fingerprint	Nmap	Identifies the services and their versions from the target (https://nmap.org/book/man-version-detection.html) while varying the attack send frequency and the duration time
All	Vulnerability Scan	Nessus	Identifies service-level vulnerabilities while varying the attack send frequency and the duration time

purpose of using a workload tool in this work was to generate valid and real requests.

Each client requested the services hosted on the honeypot server; the emulated services and their client behaviors are described in Table 1. To ensure traffic variability, each client randomly varied the requested content, according to the description shown in Table 1, and the time between the requests varied from 0 to 4 s. The variation in the time between each request and in the requested content was designed to mimic the non-modellable behavior of clients. By using this method, we could simulate a user browsing a Webpage and also sending an e-mail, for instance.

Every client generated a real and valid request for a service and received a real and valid reply from the honeypot server. Thus, all the generated background traffic was real and valid. Finally, to mitigate possible repetition in the generated traffic, we ran each of the scenarios for 30 m, a reasonable time considering the request variability shown in Table 1. To allow the scenarios to be reproduced, we logged the behavior for each client.

5.1.2. Attack traffic generation

For the attack traffic generation, we considered the taxonomy adopted by Kendall [9]. To validate the proposed method, we considered two groups of attacks as our baseline attacks: probing and DoS. The attacks and tools used and their descriptions are listed in Table 2.

Each attacker generated a specific attack type (Table 2), and to make the attacks highly variable, each attacker varied the frequency and duration during each testbed. A single machine generated each attack, allowing automatic class labeling based on the network packet source IP address. It is important to note that this approach does generate environment-specific features; e.g., on the basis only of the IP address, an anomaly-based system can identify every attack. Thus, the system being evaluated using our databases must be aware of this restriction and should not use any environment-specific features, such as the time-to-live (TTL) and IP address fields. In our opinion, this is not an issue, as these types of features are not discriminant in production environments and should not be used at all during any system evaluation for open-world production usage.

5.2. Testbed environment

The scenarios, which are described in more detail below, were composed of 100 interconnected client machines. The number of clients was established to maximize the possible client behaviors (Table 1). Each client was an Ubuntu 16.04 machine; the network traffic was dependent on the workload tool used according to the service being requested. A single honeypot server was used in each of the scenarios. The honeypot server was implemented using the Honeyd 1.5c tool, installed on an Ubuntu 16.04 machine, mimicking a vulnerable Ubuntu 14.04 server. The attacker machines ran Kali Linux version 2.0; 16 machines were used to generate the attacks, with each attacker machine generating a single type of attack (Table 2).

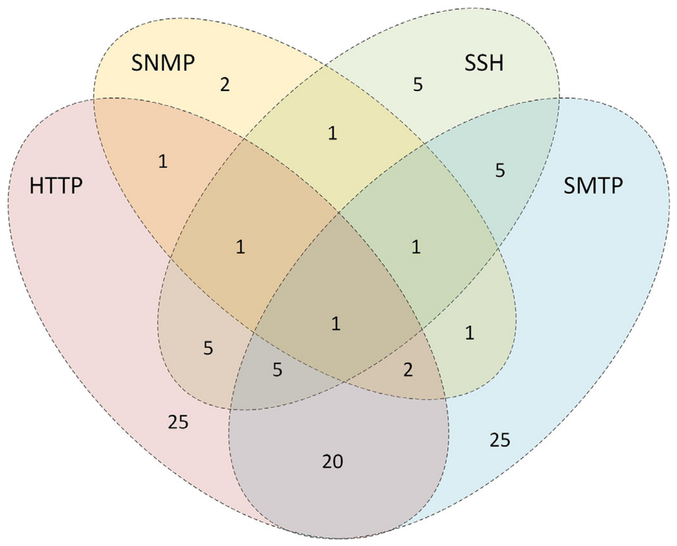


Fig. 6. Venn diagram of background service distributions among clients.

A single LAN network running at 100 Mbits/s connected the machines. The defined LAN network speed allowed the generated traffic to be recorded on a single machine without dropping packets or mirroring the traffic [7]. All legitimate requests and attacks were generated against the honeypot server (Fig. 1); the generated traffic was stored on the honeypot server.

The establishment of a single LAN network allowed the creation and replication of our proposed scenarios to be simplified. We consider that the system detection method must be tested. A detection method that presents the desirable properties of an anomaly-based detection system (Section 4.2.4) must work independently of the environment in which it was conceived. The definition of more complex scenarios [2,7] would hamper the replication of our proposed method. In the next sections, each of the created scenarios is further described.

5.2.1. Attack detection scenarios

As stated in Section 5.1.2, two attack categories were defined as our baseline attacks: probing and DoS. Thus, six scenarios were defined to generate databases to evaluate the attack detection rate (Fig. 2, Attack Detection Evaluation).

In the attack detection scenarios, each client was responsible for generating the background traffic for each client service request, as shown in the Venn diagram in Fig. 6. The overlapping circles denote the clients that generated both services. The distribution among clients and services was designed to simulate the traffic distribution described in [35].

The background network traffic remained immutable at the content request and client level. However, the attacker traffic varied according to each scenario, as described in Section 4.2.1. For each considered attack type, i.e., probing and DoS, three scenarios were created. Each scenario was run for 30 m. The attacks started at the 10th minute and lasted for 15 m (scenario time: the 10th to the 25th minute), following the attacker behavior described in Table 2. Thus, it was possible to capture the environment behavior without, with, and after the attacks. The network traffic distribution and the attacks used for each scenario are shown in Table 3.

Three levels of attack similarity were defined in the databases: network-level vulnerabilities, service-level vulnerabilities, and service-level exploitation. The first scenario was named *known*. The purpose of the *known* scenario was to generate the classifier model and to define the *known* detection rate while detecting only the known attacks. Thus, only attacks at the network level, focusing on network protocol vulnerabilities, were generated. It is important to note that the feature set used in this study (further described in Section 5.3) was specifically designed in a previous study to detect this type of attack [3]. The *similar* databases contain attacks with a similar behavior but focusing on service-level vulnerabilities. Finally, the *new* database has a new type of attack that focuses on service exploitation.

The created attack databases mimicked the behavior seen in production environments. Normally, when developing an anomaly-based NIDS, only attacks detectable by an NIDS are included in the training dataset. However, when used in production environments, the system will face a wider range of attacks. The databases were created according to the method described in Section 4.1 and were used to validate the evaluation method described in Section 4.2.

5.2.2. Background detection scenarios

The background detection rate scenarios were generated using the attacks as the baseline. Thus, two sets of attacks were used, each generated separately, resulting in different databases, allowing correct method evaluation. The sets of attacks used consisted of the probing (*known*) and DoS (*known*) attacks, shown in Table 3.

To generate the services' detection databases, the services were divided into two groups: *known* and *new* services. The *known* services served HTTP and SNMP clients and the *new* services served SMTP, NTP, and SSH clients. Each scenario was executed for 30 m; the distribution of clients followed the Venn diagram shown in Fig. 7.

Finally, to generate the content detection databases, the behavior of each client was modified. Three different behaviors were defined, divided into *known, similar*, and *new* content request behavior. The client's distribution followed the Venn diagram shown in Fig. 6, and the client's behavior for each scenario is described in Table 4. The traffic distribution for each background detection database is shown in Table 5.

5.2.3. Discussion of the databases

The proposed database creation method allowed real and valid network traffic to be generated, as the honeypot generated valid replies to each of the received requests. The event classes were automatically defined (labeled as a feature vector), to avoid manual labeling and to provide an error-free approach because of the number of packets to be evaluated. The automatic labeling was determined according to the source IP address for each network packet. This was possible because the attacker's machine generated only attack content. It is important to note that the IP address was not used as a feature value (see extracted set of features, Table 6). Thus, this knowledge does not affect the classifier model. Additionally, the use of manual class labeling or clustering techniques [36] was avoided, reducing the labeling error.

Low variability or repeated traffic occurrences were mitigated, as the client content requests, the time between each request, and the requested application were varied. The use of well-known tools allowed the databases to be updated at each new vulnerability (attack) report, as the used tool became responsible for the network traffic update to ensure the correct attack implementation. To allow the deployed scenario to be reproduced, we logged every client and attacker behavior. Finally, privacy problems did not occur, because the databases were obtained in a controlled environment and the generated network traffic did not include any sensitive data.

Using the proposed method, it was possible to create 16 databases (Tables 3 and 5), each of which was aimed to validate the common assumptions adopted in the literature [1,2], as described in Section 4.2, and to present network-specific detection rates. The following sections describe the machine learning steps of the conceived anomaly-based IDS.

Table 3
Network traffic distribution for attack detection scenarios.

Scenario	Attacks generated (Table 2)	Traffic (network packets)			Size (MB)
		Background	*Attack*	*Total*	
Probing (Known)	UDP scan, SYN scan, NULL scan, TCP connect, FIN scan, XMAS scan, and ACK scan	28,618,365	36,628	28,654,993	8.476
Probing (Similar)	OS fingerprint and service fingerprint	28,477,884	10,441	28,488,325	8.499
Probing (New)	Vulnerability scan	28,391,914	17,753	28,409,667	8.512
DoS (Known)	SYN flood, UDP flood, ICMP flood, and TCP keepalive	26,747,521	761,269	27,508,790	7.945
DoS (Similar)	SMTP flood and HTTP flood	40,278,594	26,390,723	66,669,317	12.143
DoS (New)	Vulnerability scan	27,522,317	3429	27,525,746	7.265

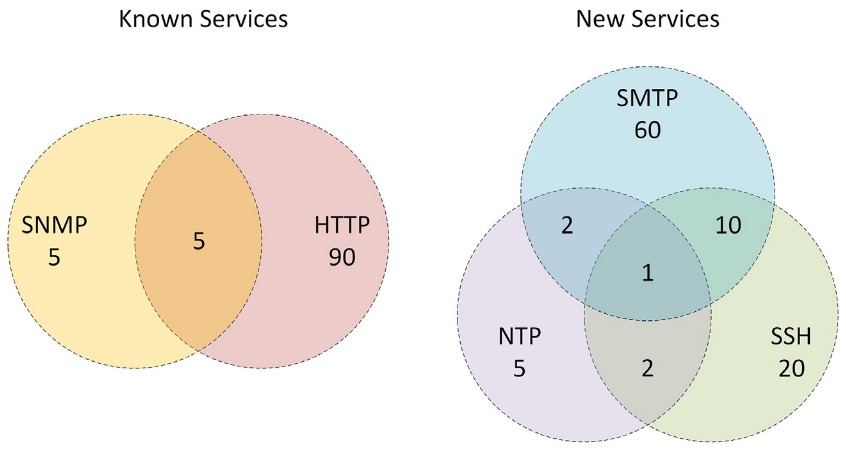


Fig. 7. Venn diagrams for the background service detection rate scenarios showing the service distributions among clients.

Table 4
Client behavior for service's content detection scenarios.

Service	Database		
	Known contents	*Similar contents*	*New contents*
HTTP	Request a Webpage from 1 to 200	Request a Webpage from 1 to 500	Request a Webpage from 501 to 1000
SMTP	Each SMTP client sends a mail with 50–400 bytes in the subject line and 100–720 bytes in the body	Each SMTP client sends a mail with 50–400 bytes in the subject line and 100–1800 bytes in the body	Each SMTP client sends a mail with 50–400 bytes in the subject line and 1801–4000 bytes in the body
SSH	Each SSH client logs in to the honeypot host and executes a random command from a list of 20 possible commands	Each SSH client logs in to the honeypot host and executes a random command from a list of 50 possible commands	Each SSH client logs in to the honeypot host and executes a random command from a list of 50 never-seen commands
SNMP	Each SNMP client operates through a predefined MIB from a predefined list of possible MIBS	Each SNMP client operates through a predefined MIB from a predefined list of possible MIBS	Each SNMP client operates through a predefined MIB from a predefined list of possible MIBS
DNS	Every name resolution is also defined in the honeypot server		

Table 5
Network traffic distribution for background detection scenarios.

Database	*Scenario*	*Generated Attacks*	*Traffic (Packets)*			*Size (MB)*
			Background	*Attack*	*Total*	
Service Detection	*Known*	*Probing (Baseline)*	6260,424	34,239	6274,663	731
	New		36,807,285	37,973	36,845,258	12,325
	Known	*DoS (Baseline)*	6874,239	753,838	7628,077	972
	New		37,273,872	812,384	38,086,256	12,738
Content Detection	*Known*	*Probing (Baseline)*	25,240,803	35,782	25,276,585	7909
	Similar		26,216,937	36,245	26,253,182	7959
	New		30,600,739	38,235	30,638,974	8905
	Known	*DoS (Baseline)*	27,376,278	746,287	28,122,565	8782
	Similar		28,241,742	784,972	29,02,6714	8932
	New		33,235,457	797,728	34,033,185	9748

5.3. Model creation process

To validate the detection system, it was necessary to build an intrusion model. This was made possible by using the created databases to extract the feature set, store the dataset, and then generate an intrusion model for each used classifier.

For each network packet read from the Network Interface Card (NIC), a set of predetermined features was extracted and sent to a classifier engine for classification. The set of features used in this work was based on our previous study [3]. Table 6 presents the 50 extracted features for each network packet, divided into three categories. All feature values were obtained by analyzing

Table 6
Extracted features set.

Category	Features
Header-based	ip_type, ip_len, ip_id, ip_offset, ip_RF, ip_DF, ip_MF, ip_proto, ip_checksum, udp_sport, udp_dport, udp_len, udp_chk, icmp_type, icmp_code, icmp_chk, tcp_sport, tcp_dport, tcp_seq, tcp_ack, tcp_ffyn, tcp_fsyn, tcp_frst, tcp_fpush, tcp_fack, tcp_furg, fr_length
Host-based	count_fr_src_dst, count_fr_dst_src, num_bytes_src_dst, num_bytes_dst_src, num_pushed_src_dst, num_pushed_dst_src, num_syn_fin_src_dst, num_syn_fin_dst_src, num_fin_src_dst, num_fin_dst_src, num_ack_src_dst, num_ack_dst_src, num_syn_src_dst, num_syn_dst_src, num_rst_src_dst, num_rst_dst_src, first_packet
Service-based	count_serv_src_dst, count_serv_dst_src, num_bytes_serv_src_dst, num_bytes_serv_dst_src, first_serv_packet, conn_status

the packet header values. The header-based category of features was extracted directly from the network packet header; e.g., *ip_type* denotes the IPV4 type field value. The host-based and service-based categories of features were extracted by analyzing the communication history between two hosts or services. In Table 6, the suffix *src_dst* indicates the traffic flow in the client to server direction, and *dst_src* that in the opposite direction. For instance, the feature *num_bytes_src_dst* counts the total number of bytes sent from the client to the server.

A 2 s time window was used to compute the time-based type of features (Table 6, host-based and service-based features), as in [10]. The feature extractor engine was implemented using the C++ language following the PCAP API using the libpcap (www.tcpdump.org) library; the implementation details were further explained in our previous paper [3]. From each network packet in the databases (Tables 3 and 5), 50 features were extracted and the feature vector was written in a separate dataset. Each feature vector entry was automatically labeled as normal or attack, on the basis of the source IP address. It is important to note that features that were scenario-specific were not considered (Table 6), e.g., TTL and IP address source or destination.

In the captured traffic, the ratio between classes was distinct (Tables 3 and 5) and most events were normal. The generation of a training and test dataset with an equal proportion of classes allowed a valid accuracy rate to be obtained without the need to verify the FP and FN rates during the model generation. Thus, a stratification process was used so that each class was equally represented in the training and validation datasets. The stratification process consisted of randomly selecting 25% of the events from the class with fewer occurrences; then, the same number of events was randomly selected from the other classes. The datasets were obtained using this stratification process, with 25% of the events being used for training, 25% for validation, and the remaining events for testing. The entire dataset was used for testing when it was not used to generate a model (Fig. 2, similar and new datasets).

For the model building process, the Weka (www.cs.waikato.ac.nz/ml/weka/) framework version 3.8.0 was used. Two classifiers were used during the evaluation tests: naïve Bayes (NB) and decision tree (DT). For the NB classifier, all numerical attributes were discretized according to the method of Fayad and Irani [37]. The C4.5 DT algorithm was used with a confidence factor of 0.25.

5.4. Model evaluation

To evaluate the proposed anomaly-based system evaluation method (Section 4.2) and multiple objective feature selection method (Section 4.3), two classifiers were used: DT and NB (note that FP denotes the number of normal instances (normal client network packets) wrongly classified as attacks, whereas FN is related to the number of attack instances wrongly classified as normal.)

To obtain the generalization capacity (Fig. 2, Generalization Evaluation), we used the publicly available DARPA1998 database [9]. Despite its well-known problems [7,10,14], DARPA1998 is still extensively used in studies in the literature (see, e.g., [8]) and provides a reasonable benchmark for research studies. The database consists of a nine-week air force environment simulation. The data of the first seven weeks are used for training and of the last two weeks for testing. For each day, DARPA1998 provides, among other files, a tcpdump file containing the network packets and a description file describing the classes for each connection.

Table 7
Traffic distribution on DARPA1998.

Category	Class	Number of packets (representativeness)
All	*Normal*	*28,426,093 (94.96%)*
DoS	*Synflood − Neptune*	*1507,319 (5.04%)*
Probing	*Port scan − Nmap*	*2211 (0.01%)*
Total		***29,935,623 (100.00%)***

For the evaluation tests, only the DARPA1998 training data were used; the feature extractor was modified to label the network packet classes according to the description file. The connection classes were divided according to Kendall's [9] taxonomy; two attack groups were considered: probing and DoS. Table 7 presents the network traffic distribution for the used classes in DARPA1998.

The rates presented for the *attack* and *normal* datasets were obtained using the test dataset (Section 5.3), which was built using the aforementioned stratification process. The following subsections present and discuss the results obtained using the traditional intrusion detection techniques and the proposed multi-objective feature selection method.

5.4.1. Traditional model building process

The traditional model building process was divided into two groups: in one (*selection*) the traditional feature selection was performed and in the second (*no-selection*) it was not.

The *selection* group relied on the traditional feature selection method, as described in [3,38,39]. For this purpose, a wrapper-based GA feature selection method was used, the objective of which was to increase the accuracy in the validation dataset. The GA was used with 100 generations and 100 populations for each generation, a mutation probability of 3.3%, and a 60% crossover probability. The *no-selection* group used the 50 extracted features (Table 6) during the model building process. The classifiers were developed using the Weka framework.

To perform the traditional model building process, we considered the approach normally used in studies in the literature [3]. The *no-selection* and *selection* groups were trained, validated, and tested using the *known* datasets (Fig. 2, *Known Attacks, Known Services,* and *Known Services' Content*), whereas the generated models were evaluated using the remaining datasets (Fig. 2). The presented rates were obtained from the test dataset, when the dataset was also used for training (Fig. 2, *Known datasets*), and from the entire dataset, when the dataset was used only for the tests (Fig. 2, *Similar, New*, and *Publicly Available* datasets). The obtained attack detection rates are presented in Table 8.

The DT and NB classifiers could obtain a reasonable high accuracy rate in both the *Probing* and the *DoS attack*^{known} datasets.

Table 8
Rates obtained for attack detection scenarios (Table 3).

Attacks	*Classifier*	*Model Building Method*	Dataset								
			attack^{known}			*attack^{similar}*			*attack^{new}*		
			Accuracy (%)	*FP (%)*	*FN (%)*	*Accuracy (%)*	*FP (%)*	*FN (%)*	*Accuracy (%)*	*FP (%)*	*FN (%)*
Probing	*Decision Tree*	*no-selection*	***99.99***	*0.02*	*0.00*	***98.62***	*0.04*	*2.72*	***64.66***	*0.09*	*70.59*
		selection	***99.99***	*0.00*	*0.02*	***93.70***	*0.04*	*12.57*	***53.30***	*0.00*	*91.39*
		multi-objective (attack)	***99.82***	*0.23*	*0.12*	***99.41***	*0.11*	*1.07*	***99.76***	*0.09*	*0.38*
		multi-objective (normal)	***99.93***	*0.01*	*0.13*	***99.27***	*0.04*	*1.42*	***74.56***	*0.05*	*50.83*
		multi-objective (generalization)	***99.88***	*0.15*	*0.09*	***99.66***	*0.11*	*0.57*	***96.34***	*0.16*	*7.17*
		multi-objective (all)	***99.87***	*0.13*	*0.13*	***99.29***	*0.04*	*1.38*	***96.12***	*0.02*	*7.73*
	Naïve Bayes	*no-selection*	***99.75***	*0.27*	*0.22*	***99.04***	*0.15*	*1.76*	***57.38***	*0.18*	*85.06*
		selection	***99.99***	*0.01*	*0.00*	***99.37***	*0.00*	*1.26*	***65.72***	*0.02*	*68.54*
		multi-objective (attack)	***99.61***	*0.72*	*0.07*	***99.35***	*0.46*	*0.84*	***98.61***	*0.38*	*2.39*
		multi-objective (normal)	***99.88***	*0.12*	*0.12*	***98.80***	*0.19*	*2.22*	***62.18***	*0.14*	*75.51*
		multi-objective (generalization)	***99.47***	*0.66*	*0.41*	***96.90***	*2.68*	*3.52*	***90.92***	*2.32*	*15.84*
		multi-objective (all)	***99.67***	*0.45*	*0.21*	***97.76***	*1.26*	*3.22*	***92.03***	*1.13*	*14.80*
DoS	*Decision Tree*	*no-selection*	***99.97***	*0.01*	*0.06*	***99.95***	*0.03*	*0.08*	***75.61***	*0.00*	*48.77*
		selection	***99.99***	*0.00*	*0.03*	***99.96***	*0.03*	*0.06*	***51.11***	*0.00*	*97.78*
		multi-objective (attack)	***99.99***	*0.01*	*0.01*	***99.91***	*0.15*	*0.03*	***92.59***	*0.23*	*14.59*
		multi-objective (normal)	***99.98***	*0.01*	*0.02*	***99.90***	*0.14*	*0.05*	***79.35***	*0.00*	*41.31*
		multi-objective (generalization)	***99.98***	*0.01*	*0.03*	***99.94***	*0.03*	*0.09*	***74.27***	*0.00*	*51.46*
		multi-objective (all)	***99.98***	***0.01***	*0.02*	***99.93***	*0.04*	*0.10*	***90.08***	*0.00*	*19.84*
	Naïve Bayes	*no-selection*	***99.90***	*0.09*	*0.11*	***97.76***	*0.79*	*3.69*	***57.29***	*0.00*	*85.41*
		selection	***99.95***	*0.01*	*0.05*	***98.95***	*0.09*	*2.00*	***50.70***	*0.00*	*98.60*
		multi-objective (attack)	***99.92***	*0.00*	*0.15*	***98.10***	*1.96*	*1.83*	***88.68***	*0.70*	*21.94*
		multi-objective (normal)	***99.35***	*1.17*	*0.13*	***98.95***	*0.09*	*2.00*	***51.92***	*0.00*	*96.15*
		multi-objective (generalization)	***98.94***	*0.10*	*2.06*	***98.03***	*1.39*	*2.55*	***54.78***	*0.00*	*90.43*
		multi-objective (all)	***99.93***	*0.00*	*0.13*	***98.65***	*1.73*	*0.97*	***81.33***	*0.35*	*36.99*

The average accuracy and FN rates for the known attacks for both *Probing* and *DoS* were 99.87% and 99.93% for the *no-selection* group and 99.99% and 99.97% for the *selection* group, respectively. The traditional feature selection process (*selection*) improved the classification accuracy for known attacks by an average of 0.08%.

However, in the similar attack datasets, it was possible to observe an increase in the FN rates. The worst classifier was DT with the *selection* method, which showed a 12.57% FN rate on the *Probing* database, whereas NB with the *no-selection* method showed a 3.69% FN rate. On average, the FN rate increased by 3.97% and 2.06% with the *selection* and *no-selection* approach, respectively. Thereby, it can be stated that the current approaches in the literature can detect similar attacks with a small increase in the FN rate, 3.01% on average. In most cases, the *selection* approach decreased the FN rate for detecting similar attacks; the only case where the FN rate increased was that of DT on the *Probing* database, which showed a 9.85% rate. The FP rates remained almost unchanged, with an average increase of 0.09% with both model building methods; thus, it is possible to note that the used services (Table 1) still present the same or similar behavior under different attacks. Finally, for detecting new attacks, the FN rates significantly increased.

The best FN rate was obtained by NB with the *selection* method with a 68.54% FN rate on the *Probing* dataset, whereas a 48.77% FN rate was obtained by DT with the *no-selection* method on the *DoS* dataset. The results show the inability of machine learning methods to detect new attacks in the evaluated scenarios. Neither of the classifiers could maintain the obtained rates during the model testing on known attacks. The anomaly-based assumption for detecting new attacks was not evidenced during the evaluation tests, where the traditional detection approaches were used: the FN rate as compared to that of the testing phase, was increased by 72.46% and 89.05%, on average, for the *no-selection* and *selection* methods, respectively.

The obtained background detection rates are presented in Table 9. Each classifier and model building method could detect known services and known services' content, reaching an average FP rate of 0.13% and 0.08%, respectively. The *selection* method improved the FP rate by 0.25% on average for known services' content and by 0.14% for known services.

For detecting new services, the FP rate greatly increased; in general, the *selection* method increased the FP rate significantly: 20.82% on average against 6.03% with the *no-selection* method. However, for detecting similar and new services' content, the *selection* method showed an average FP rate of 0.68% for both similar and new services' content, whereas the *no-selection* method showed an FP rate of 1.50% and 3.06% for similar and new services' content, respectively.

Finally, Table 10 shows the generalization evaluation performed on DARPA1998. A significant increase in the FP and FN rates can be observed when the model was used in a different scenario. Several observations can be made from Tables 8–10 regarding the traditional model building methods:

- Both classifiers, regardless of the model building method used, could detect known events (attacks, services, and services' content). The worst detection rates were 99.75% for known attacks, 99.67% for known services' content, and 99.98% for known services.
- None of the classifiers could maintain its obtained accuracy on the model building dataset for detecting new attacks. The anomaly-based assumption for detecting new attacks was not evidenced when the machine learning technique was used.
- The traditional machine learning technique was able to detect similar attacks (Table 8; 1.97% and 3.95% FP rate increase for the *no-selection* and *selection* methods, respectively), making it a viable approach for detecting possible intrusion attempts, provided that the attacks present a similar behavior.
- In general, there was an FP increase for detecting new services (Table 9; 13.42% average FP rate); however, the accuracy loss was less than that observed for detecting new attacks (Table 7; 80.77% average FN rate).
- A small increase in the FP rate was evidenced for detecting new services' content (Table 9; 2.80% and 0.67% FP rate increase for the *no-selection* and *selection* methods, respectively); however,

Table 9
Rates obtained for normal detection scenarios (table 5).

Attacks	*Classifier*	*Model building method*	*Dataset*														
			normal$^{known\ content}$			normal$^{similar\ content}$			normal$^{new\ content}$			normal$^{known\ service}$			normal$^{new\ service}$		
			Accuracy (%)	*FP (%)*	*FN (%)*	*Accuracy (%)*	*FP (%)*	*FN (%)*	*Accuracy (%)*	*FP (%)*	*FN (%)*	*Accuracy (%)*	*FP (%)*	*FN (%)*	*Accuracy (%)*	*FP (%)*	*FN (%)*
Probing	*Decision Tree*	*no-selection*	***99.94***	*0.12*	*0.00*	***99.97***	*0.05*	*0.00*	***99.98***	*0.04*	*0.00*	***99.96***	*0.05*	*0.02*	***92.93***	*14.12*	*0.02*
		selection	***100.00***	*0.00*	*0.00*	***98.89***	*2.22*	*0.00*	***98.74***	*2.51*	*0.00*	***99.99***	*0.01*	*0.00*	***99.34***	*1.31*	*0.00*
		multi-objective (attack)	***99.90***	*0.20*	*0.00*	***99.81***	*0.37*	*0.00*	***99.73***	*0.54*	*0.00*	***99.91***	*0.14*	*0.03*	***98.50***	*2.98*	*0.03*
		multi-objective (normal)	***99.99***	*0.01*	*0.00*	***99.99***	*0.01*	*0.00*	***99.99***	*0.01*	*0.00*	***99.97***	*0.01*	*0.05*	***99.92***	*0.11*	*0.05*
		multi-objective (general)	***99.92***	*0.07*	*0.09*	***98.85***	*2.22*	*0.09*	***97.30***	*5.31*	*0.09*	***99.86***	*0.23*	*0.05*	***86.41***	*27.13*	*0.05*
		multi-objective (all)	***99.95***	*0.08*	*0.02*	***99.93***	*0.11*	*0.02*	***99.96***	*0.07*	*0.02*	***99.94***	*0.05*	*0.02*	***99.84***	*0.27*	*0.05*
	Naïve Bayes	*no-selection*	***99.67***	*0.52*	*0.14*	***98.31***	*3.23*	*0.14*	***96.90***	*6.05*	*0.14*	***99.83***	*0.31*	*0.03*	***96.96***	*6.05*	*0.03*
		selection	***99.99***	*0.02*	*0.00*	***99.81***	*0.37*	*0.00*	***96.96***	*0.08*	*0.00*	***99.99***	*0.01*	*0.00*	***79.43***	*41.15*	*0.00*
		multi-objective (attack)	***99.54***	*0.93*	*0.00*	***98.08***	*3.93*	*0.00*	***96.55***	*6.90*	*0.00*	***99.37***	*1.27*	*0.00*	***96.06***	*7.87*	*0.00*
		multi-objective (normal)	***99.94***	*0.04*	*0.08*	***99.92***	*0.09*	*0.08*	***99.91***	*0.11*	*0.08*	***99.97***	*0.02*	*0.03*	***99.46***	*1.05*	*0.03*
		multi-objective (general)	***99.56***	*0.68*	*0.22*	***97.88***	*4.03*	*0.22*	***95.02***	*9.74*	*0.22*	***98.73***	*2.41*	*0.13*	***98.88***	*6.12*	*0.13*
		multi-objective (all)	***99.61***	*0.64*	*0.13*	***98.13***	*3.60*	*0.13*	***96.55***	*6.77*	*0.13*	***98.94***	*2.05*	*0.08*	***96.50***	*6.92*	*0.08*
DoS	*Decision Tree*	*no-selection*	***99.99***	*0.03*	*0.00*	***99.98***	*0.04*	*0.00*	***99.97***	*0.05*	*0.00*	***98.98***	*0.02*	*0.02*	***99.78***	*0.43*	*0.02*
		selection	***100.00***	*0.01*	*0.00*	***99.97***	*0.07*	*0.00*	***99.98***	*0.04*	*0.00*	***100.00***	*0.01*	*0.00*	***98.27***	*3.46*	*0.00*
		multi-objective (attack)	***99.96***	*0.03*	*0.04*	***99.94***	*0.08*	*0.04*	***99.90***	*0.15*	*0.04*	***99.93***	*0.09*	*0.04*	***93.33***	*13.29*	*0.04*
		multi-objective (normal)	***99.98***	*0.03*	*0.02*	***99.99***	*0.01*	*0.02*	***99.98***	*0.02*	*0.02*	***99.99***	*0.00*	*0.02*	***99.98***	*0.02*	*0.02*
		multi-objective (general)	***99.96***	*0.06*	*0.02*	***99.97***	*0.05*	*0.02*	***99.98***	*0.03*	*0.02*	***99.97***	*0.05*	*0.01*	***99.98***	*0.04*	*0.01*
		multi-objective (all)	***99.96***	*0.03*	*0.05*	***99.92***	*0.11*	*0.05*	***99.91***	*0.13*	*0.05*	***99.94***	*0.07*	*0.05*	***99.92***	*0.12*	*0.05*
	Naïve Bayes	*no-selection*	***99.50***	*0.36*	*0.65*	***98.33***	*2.69*	*0.65*	***96.63***	*6.08*	*0.65*	***99.64***	*0.20*	*0.51*	***97.99***	*3.50*	*0.51*
		selection	***100.00***	*0.00*	*0.00*	***99.98***	*0.04*	*0.00*	***99.95***	*0.09*	*0.00*	***100.00***	*0.00*	*0.00*	***81.30***	*37.35*	*0.00*
		multi-objective (attack)	***98.91***	*0.49*	*1.69*	***98.07***	*2.16*	*1.69*	***98.71***	*0.89*	*1.69*	***99.35***	*0.29*	*1.02*	***92.37***	*14.23*	*1.02*
		multi-objective (normal)	***99.99***	*0.00*	*0.03*	***99.98***	*0.02*	*0.03*	***99.98***	*0.02*	*0.03*	***99.99***	*0.00*	*0.03*	***99.82***	*0.34*	*0.03*
		multi-objective (general)	***98.93***	*0.04*	*2.11*	***98.32***	*1.26*	*2.11*	***97.20***	*3.48*	*2.11*	***98.93***	*0.02*	*2.12*	***91.91***	*14.06*	*2.12*
		multi-objective (all)	***99.86***	*1.20*	*0.08*	***98.98***	*1.95*	*0.08*	***98.09***	*3.74*	*0.08*	***99.77***	*0.13*	*0.32*	***98.73***	*1.74*	*0.32*

Table 10
Rates obtained for generalization evaluation.

Attacks	*Classifier*	*Model building method*	*Accuracy (%)*	*FP (%)*	*FN (%)*
Probing	*Decision Tree*	*no-selection*	***86.97***	*15.20*	*10.85*
		selection	***90.38***	*8.86*	*10.36*
		multi-objective (attack)	***81.55***	*31.80*	*5.11*
		multi-objective (normal)	***86.79***	*15.78*	*10.63*
		multi-objective (general)	***98.42***	*2.89*	*0.27*
		multi-objective (all)	***96.25***	*7.24*	*0.27*
	Naïve Bayes	*no-selection*	***75.93***	*48.12*	*0.00*
		selection	***78.56***	*33.29*	*9.59*
		multi-objective (attack)	***83.77***	*32.43*	*0.00*
		multi-objective (normal)	***68.75***	*7.91*	*54.59*
		multi-objective (general)	***97.29***	*5.43*	*0.00*
		multi-objective (all)	***96.43***	*7.15*	*0.00*
DoS	*Decision Tree*	*no-selection*	***38.98***	*30.47*	*91.57*
		selection	***99.41***	*0.00*	*1.18*
		multi-objective (attack)	***94.56***	*10.89*	*0.00*
		multi-objective (normal)	***85.23***	*29.54*	*0.00*
		multi-objective (general)	***99.90***	*0.20*	*0.00*
		multi-objective (all)	***99.36***	*1.29*	*0.00*
	Naïve Bayes	*no-selection*	***82.71***	*34.56*	*0.01*
		selection	***84.26***	*31.19*	*0.29*
		multi-objective (attack)	***81.82***	*29.28*	*7.09*
		multi-objective (normal)	***93.53***	*12.77*	*0.05*
		multi-objective (general)	***99.66***	*0.52*	*0.16*
		multi-objective (all)	***95.23***	*8.57*	*0.97*

in most cases, the classifiers were able to correctly distinguish the classes.

- When using the obtained classifiers in a different environment (Table 10), the detection accuracy significantly decreased, even for detecting known attacks; in most cases, the model became scenario-dependent.

The proposed evaluation method allowed the common assumptions presented in the literature to be verified. Moreover, it allowed rich intrusion detection properties to be obtained from the intrusion detection, which helps experts determine whether their systems are reliable for open-world usage or not.

During the evaluation tests using the traditional machine learning model building techniques it was observed that, when a classifier faced known events, it presented a reasonable accuracy rate. The results showed a decrease in accuracy when a classifier faced similar events; the effect on accuracy further increased when a classifier faced new attacker and client behaviors. The next subsection evaluates our proposed multi-objective feature selection method (Section 4.3).

5.4.2. Multi-objective feature selection

In our proposed multi-objective feature selection method for intrusion detection systems (Section 4.3), we used the well-known

Table 11
Rates obtained for each considered objective.

Attacks	*Classifier*	*Model building method*	$attack_{rate}$	$normal_{rate}$	gen_{rate}
Probing	*Decision Tree*	*no-selection*	*87.76*	*98.56*	*86.97*
		selection	*82.33*	*99.39*	*90.38*
		multi-objective (attack)	***99.66***	*99.57*	*81.55*
		multi-objective (normal)	*91.25*	***99.97***	*86.79*
		multi-objective (general)	*98.63*	*96.47*	***98.42***
		multi-objective (all)	*98.43*	*99.92*	*96.25*
	Naïve Bayes	*no-selection*	*85.39*	*98.33*	*75.93*
		Selection	*88.36*	*95.24*	*78.56*
		multi-objective (attack)	***99.19***	*97.92*	*83.77*
		multi-objective (normal)	*86.95*	***99.84***	*68.75*
		multi-objective (general)	*95.76*	*98.01*	***97.29***
		multi-objective (all)	*96.49*	*97.95*	*96.43*
DoS	*Decision Tree*	*no-selection*	*91.84*	*99.74*	*38.98*
		Selection	*83.69*	*99.64*	*99.41*
		multi-objective (attack)	***97.50***	*98.61*	*94.56*
		multi-objective (normal)	*93.08*	***99.98***	*85.23*
		multi-objective (general)	*91.40*	*99.97*	***99.90***
		multi-objective (all)	*96.66*	*99.93*	*99.36*
	Naïve Bayes	*no-selection*	*84.98*	*98.42*	*82.71*
		Selection	*83.20*	*96.25*	*84.26*
		multi-objective (attack)	***95.57***	*97.48*	*81.82*
		multi-objective (normal)	*83.41*	***99.95***	*93.53*
		multi-objective (general)	*83.92*	*97.06*	***99.66***
		multi-objective (all)	*93.30*	*99.09*	*95.23*

NSGA-II [18] algorithm. As previously described (Section 4.3), we considered three objectives during our model building process: the $attack_{rate}$ (Eq. (1)), $normal_{rate}$ (Eq. (4)), and $generalization_{rate}$ (Eq. (5)). NSGA-II operates by minimizing the objectives; thus, for the purposes of our tests, we considered the obtained error rate in our evaluation tests. The same set of parameters used by the traditional feature selection process was used (Section 5.4.1): 100 generations and 100 populations for each generation, a mutation probability of 3.3%, and a 60% crossover probability.

As stated in Section 4.3, the desired objective must be defined according to the administrator's needs. Thus, for test purposes, four operation points were chosen: *attack, normal, generalization*, and *all*. Each chosen operating point presented the lowest error rate related to its objective: *attack* presented the lowest $attack_{rate}$ (Eq. (1)) error rate, *normal* presented the lowest $normal_{rate}$ (Eq. (4)) error rate, *generalization* presented the lowest $generalization_{rate}$ (Eq. (5)) error rate, and finally, *all* presented the lowest error rate considering all objectives.

The obtained objective rates are presented in Table 11. The proposed multi-objective feature selection achieved the best results in all cases for detecting its considered objective.

The *multi-objective (attack)* operation point improved the $attack_{rate}$ in all cases. As compared to the traditional *selection* method, it improved the $attack_{rate}$ by 17.33% and 10.83% for the DT and NB classifiers on the *Probing dataset,* respectively, while improving it by 13.81% and 12.37% for the DT and NB classifiers on the *DoS* dataset, respectively. On average, the *multi-objective (attack)* improved the $attack_{rate}$ accuracy by 10.49% and 13.59% for the *no-selection* and *selection* methods, respectively. As compared to the other operation points, *multi-objective (attack)* improved the $attack_{rate}$ accuracy by 5.54% on average.

In the individual attack detection accuracy (Table 7), a significant improvement can be observed. The *multi-objective (attack)* operating point significantly improved the detection of similar and new attacks. On average, it improved by 0.77% and 0.27% for similar attacks as compared to the traditional model building methods and the other operating points, respectively. For the detection of new attacks, the *multi-objective (attack)* operating point enabled the detection of new attacks in most cases, improving the detection rate of new attacks by 35.44% as compared to the traditional model building methods and by 16.25%, on average, as compared to the other operating points.

Finally, it is possible to note a tradeoff between the $attack_{rate}$ objective and the other objectives. In most cases, the *multi-objective (attack)* operation point reduced the accuracy of the other objectives; as compared to the other operation points, the only case where the objective was improved was for the DT classifier on the *Probing* dataset, showing an improvement of 0.78% on average for the normal objective. On average, the tradeoff between objectives when the attack objective was considered was −0.62% for normal detection and −7.65% for the generalization objective.

The *multi-objective (normal)* operation point slightly improved the $normal_{rate}$ objective as compared to the other techniques in all cases. The *multi-objective (normal)* operation point improved the detection of normal events on average by 1.74% and 1.44% as compared to the traditional model building methods and the other operating points, respectively. These small accuracy improvements occurred as a result of the models' capacity to detect normal events with little or no effect on accuracy (Section 5.4.1). It is possible to note a significant tradeoff between the detection of normal events and attack events, and generalization. On average, as compared to the other operating points, when the *multi-objective (normal)* operation point was considered the detection of attacks was reduced by 6.87%, whereas the model generalization capacity decreased by 10.11%.

The *multi-objective (generalization)* operation point significantly increased the $generalization_{rate}$ in all scenarios. As compared to the traditional model building techniques, the *multi-objective (generalization)* operation point presented, on average, a 19.17% higher accuracy rate, while it was increased, on average, by 10.21% as compared to the other operation points. Regarding the *multi-objective (generalization)* operation point tradeoff, on average, it was evidenced that there was a decrease in the attack detection rate of 6.48% and 8.35%, whereas for the normal detection, there were an increase of 0.32% and a decrease of 0.99% for the traditional model building methods and the other operating points, respectively. Thus, to provide generalization, a significant tradeoff between attack and normal detection rates is required. However, the most important point pertaining to generalization was that an old benchmark database was used, because if the generalization

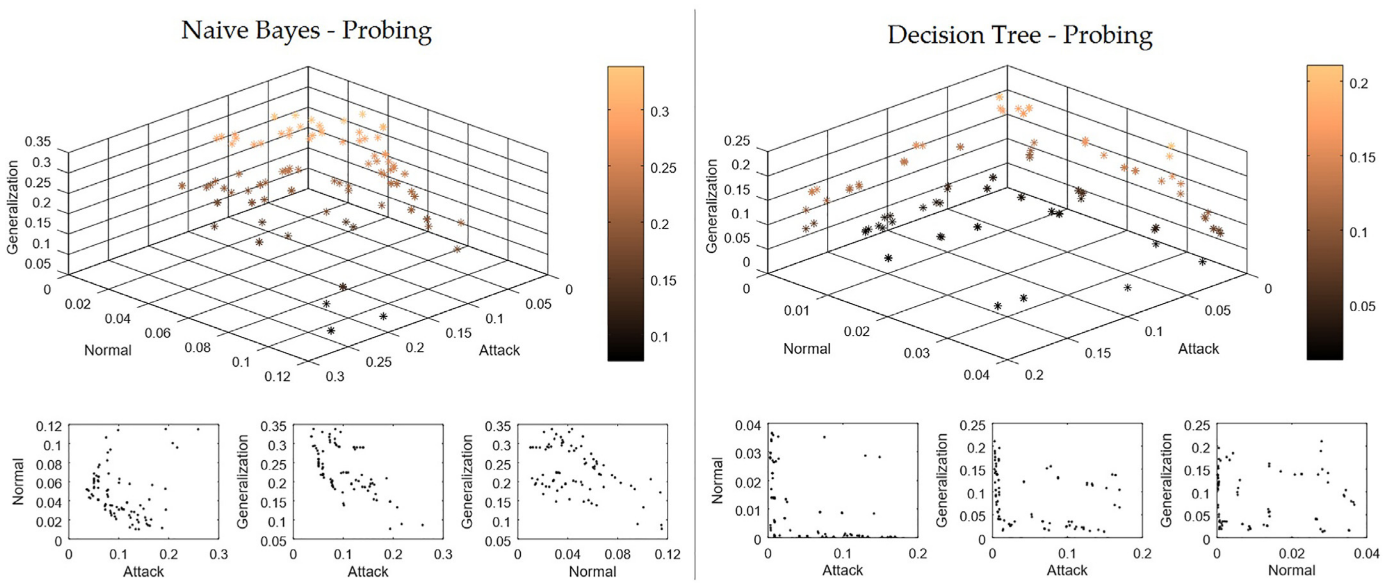


Fig. 8. Multi-objective operation points for probing attacks. The operation points are shown in terms of the objective error rate; the operation points for each objective are chosen according to their lowest error rate point.

rate results are good, it means that the proposed database is equivalent to the old one. Therefore, we can use an updated database to test recent attacks.

Finally, the *multi-objective (all)* operation point was aimed to improve all the considered objectives. When considering all the objectives, an improvement of 9.49% was shown: the average detection rate was 97.42%, whereas the average detection rate for all the objectives using the traditional model building methods was 87.93%; when the other operating points were considered, the average detection rate was 93.68%.

Fig. 8 shows the operation points through our method for the detection of probing attacks. It can be noticed that the generalization capacity increases while the attack detection rate decreases. However, the detection of normal events does not significantly decrease the system generalization capacity. Similar behavior was observed for the detection of DoS attacks.

6. Conclusion

The constantly increasing number of network threats demands security approaches that are reliable. Thus, over the last few years, a great number of research studies has been conducted on anomaly-based intrusion detection; however, despite the promising results, this technique is hardly used in production. To improve this situation, this paper presented three contributions for developing reliable anomaly-based intrusion detection systems. Initially, we proposed a new tool-based intrusion database creation method that is aimed to produce databases that can easily be updated, reproduce real and valid traffic, are representative, and are publicly available. Through the proposed intrusion database creation method, a new evaluation scheme specific to the machine learning intrusion detection field was presented. This scheme allowed each of the common assumptions in the literature to be validated, such as that new events and new services are detected. Finally, to provide a reliable anomaly-based intrusion detection system, we presented and evaluated a multiple objective feature selection method. The evaluation approach allows a system administrator to establish the real capacity of a system for detecting each of the common properties in any production environment.

All the data presented in this paper are publicly available for download at: https://secplab.ppgia.pucpr.br/trabid.

Acknowledgment

The authors would like to thank Intel Labs Univ. Research Office and the Brazilian National Council for Scientific and Technological Development (grant 440850/2013-4) for the financial support.

References

[1] C. Gates, C. Taylor, Challenging the anomaly detection paradigm: a provocative discussion, in: Proc. 2006 Work. New Secur. Paradig., 2007, pp. 21–29, doi:10.1145/1278940.1278945.

[2] R. Sommer, V. Paxson, Outside the closed world: on using machine learning for network intrusion detection, in: Proc. IEEE Symp. Secur. Priv., 2010, pp. 305–316, doi:10.1109/SP.2010.25.

[3] E. Viegas, A. Santin, A. França, R. Jasinski, V. Pedroni, L. Oliveira, Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems, IEEE Trans. Comput. (2017) 163–177, doi:10.1109/TC.2016.2560839.

[4] V. Paxson, S. Floyd, Wide-area traffic: the failure of Poisson modeling, *IEEE/ACM Transactions on Networking* 3 (1995) 226–244, doi:10.1109/90.392383.

[5] S. Axelsson, The base-rate fallacy and the difficulty of intrusion detection, ACM Trans. Inf. Syst. Secur. 3 (2000) 186–205, doi:10.1145/357830.357849.

[6] M.V Mahoney, P.K. Chan, An analysis of the 1999 DARPA/Lincoln laboratory evaluation data for network anomaly detection, Proc. Sixth Int. Symp. Recent Adv. Intrusion Detect. 2820 (2003) 220–237, doi:10.1007/b13476.

[7] A. Shiravi, H. Shiravi, M. Tavallaee, A. a. Ghorbani, Toward developing a systematic approach to generate benchmark datasets for intrusion detection, Comput. Secur. 31 (2012) 357–374, doi:10.1016/j.cose.2011.12.012.

[8] C.F. Tsai, Y.F. Hsu, C.Y. Lin, W.Y. Lin, Intrusion detection by machine learning: a review, Expert Syst. Appl. 36 (2009) 11994–12000, doi:10.1016/j.eswa.2009.05.029.

[9] K. Kendall, A database of computer attacks for the evaluation of intrusion detection systems S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1999.

[10] J. McHugh, Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, ACM Trans. Inf. Syst. Secur. 3 (2000) 262–294, doi:10.1145/382912.382923.

[11] B. Wang, Y. Zheng, W. Lou, Y.T. Hou, DDoS attack protection in the era of cloud computing and software-defined networking, Comput. Netw. 81 (2015) 308–319, doi:10.1016/j.comnet.2015.02.026.

[12] A. Jamdagni, Z. Tan, X. He, P. Nanda, R.P. Liu, RePIDS: a multi tier real-time payload-based intrusion detection system, Comput. Netw. 57 (2013) 811–824, doi:10.1016/j.comnet.2012.10.002.

[13] V. Bolón-Canedo, N. Sánchez-Maroño, a. Alonso-Betanzos, Feature selection and classification in multiple class datasets: an application to KDD Cup 99 dataset, Expert Syst. Appl. 38 (2011) 5947–5957, doi:10.1016/j.eswa.2010.11.028.

[14] M. Tavallaee, E. Bagheri, W. Lu, A. a. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: IEEE Symp. Comput. Intell. Secur. Def. Appl. CISDA 2009., 2009, pp. 1–6, doi:10.1109/CISDA.2009.5356528.

[15] G. Creech, J. Hu, Generation of a new IDS test dataset: time to retire the KDD collection, in: IEEE Wirel. Commun. Netw. Conf. WCNC., 2013, pp. 4487–4492, doi:10.1109/WCNC.2013.6555301.

[16] T.T.T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, Commun. Surv. Tut. IEEE. 10 (2008) 56–76, doi:10.1109/SURV.2008.080406.

[17] H.M. Ammari, On the energy-delay trade-off in geographic forwarding in always-on wireless sensor networks: a multi-objective optimization problem, Comput. Netw. 57 (2013) 1913–1935, doi:10.1016/j.comnet.2013.03.009.

[18] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2002) 182–197, doi:10.1109/4235.996017.

[19] M.H. Bhuyan, D.K. Bhattacharyya, J.K. Kalita, Towards generating real-life datasets for network intrusion detection, Int. J. Netw. Secur. 17 (2015) 683–701.

[20] D.E. Denning, An intrusion-detection model, Proc. - IEEE Symp. Secur. Priv. (2012) 118–131, doi:10.1109/SP.1986.10010.

[21] J. Peng, K.K.R. Choo, H. Ashman, User profiling in intrusion detection: a review, J. Netw. Comput. Appl. 72 (2016) 14–27, doi:10.1016/j.jnca.2016.06.012.

[22] A.I. Abubakar, H. Chiroma, S.A. Muaz, L.B. Ila, A review of the advances in cyber security benchmark datasets for evaluating data-driven based intrusion detection systems, Procedia Comput. Sci. 62 (2015) 221–227, doi:10.1016/j.procs.2015.08.443.

[23] D. Canali, M. Cova, G. Vigna, C. Kruegel, Prophiler: A fast filter for the large-scale detection of malicious web pages categories and subject descriptors, in: Proc. Int. World Wide Web Conf., 2011, pp. 197–206, doi:10.1145/1963405.1963436.

[24] J. Yang, Y. Qiao, X. Zhang, H. He, F. Liu, G. Cheng, Characterizing user behavior in mobile internet, IEEE Trans. Emerg. Top. Comput. 3 (2015) 95–106, doi:10.1109/TETC.2014.2381512.

[25] A.D. Kent, L.M. Liebrock, J.C. Neil, Authentication graphs: analyzing user behavior within an enterprise network, Comput. Secur. 48 (2015) 150–166, doi:10.1016/j.cose.2014.09.001.

[26] N. Munaiah, A. Meneely, R. Wilson, B. Short, Are intrusion detection studies evaluated consistently? A systematic literature review, Rochester Institute of Technology, Technical Report (2016). http://scholarworks.rit.edu/article/1810 (accessed 17.08.17).

[27] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, M.A. Zissman, Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation, in: Proc. DARPA Inf. Surviv. Conf. Expo. DISCEX'00. 2, 2000, doi:10.1109/DISCEX.2000.821506.

[28] T. Probst, E. Alata, M. Kaaniche, V. Nicomette, Automated evaluation of network intrusion detection systems in IaaS Clouds, in: 2015 11th Eur. Dependable Comput. Conf., 2015, pp. 49–60, doi:10.1109/EDCC.2015.10.

[29] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, B.D. Payne, Evaluating computer intrusion detection systems: a survey of common practices, ACM Comput. Surv. 48 (2015) 1–41, doi:10.1145/2808691.

[30] J. Jabez, B. Muthukumar, Intrusion Detection System (IDS): anomaly detection using outlier detection approach, Procedia Comput. Sci. 48 (2015) 338–346, doi:10.1016/j.procs.2015.04.191.

[31] I. Syarif, A. Prugel-Bennett, G. Wills, Data mining approaches for network intrusion detection: from dimensionality reduction to misuse and anomaly detection, J. Inf. Technol. Rev. 3 (2012) 70–83.

[32] S.Y. Ji, B.K. Jeong, S. Choi, D.H. Jeong, A multi-level intrusion detection method for abnormal network behaviors, J. Netw. Comput. Appl. 62 (2016) 9–17, doi:10.1016/j.jnca.2015.12.004.

[33] C.K. Olivo, A.O. Santin, L.S. Oliveira, Obtaining the threat model for e-mail phishing, Appl. Soft Comput. J. 13 (2013) 4841–4848, doi:10.1016/j.asoc.2011.06.016.

[34] D. Darmon, J. Sylvester, M. Girvan, W. Rand, Understanding the predictive power of computational mechanics and echo state networks in social media, Technical Report (2013). arXiv: https://arxiv.org/abs/1306.6111v2 (accessed 17.08.17)

[35] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 White Paper, 2017, Document ID:1454457600805266.

[36] T. Heimann, P. Mountney, M. John, R. Ionasec, Learning Without Labeling: Domain Adaptation For Ultrasound Transducer Localization, LNCS 8151 (2013) 49–56, doi:10.1007/978-3-642-40760-4_7.

[37] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, in: Proc. Int. Jt. Conf. Uncertain. AI., 1993, pp. 1022–1027. http://trs-new.jpl.nasa.gov/dspace/handle/2014/35171.

[38] M.S. Hoque, M.A. Mukit, M.A.N. Bikas, M. Sazzadul Hoque, An implementation of intrusion detection system using genetic algorithm, Int. J. Netw. Secur. Its Appl. 4 (2012) 109–120, doi:10.4156/jcit.vol4.issue1.janakiraman.

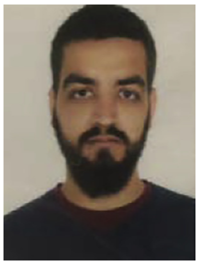
[39] D.S. Kim, H. Nguyen, J.S. Park, Genetic algorithm to improve SVM based network intrusion detection system, 19th Int. Conf. Adv. Inf. Netw. Appl. 2005, 1155–1158. doi:10.1109/AINA.2005.191.

[40] E. De, E. De, A. Ortiz, J. Ortega, A. Martínez-Álvarez, Feature selection by multi-objective optimisation: application to network anomaly detection by hierarchical self-organising maps, Knowl. Based Syst. (2014), doi:10.1016/j.knosys.2014.08.013.

[41] U. Shaukat, Z. Anwar, A fast and scalable technique for constructing multicast routing trees with optimized quality of service using a firefly based genetic algorithm, Multimed. Tools Appl. 75 (2016) 2275–2301, doi:10.1007/s11042-014-2405-4.

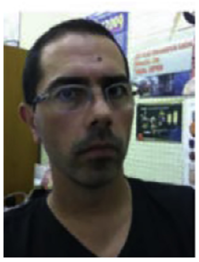
[42] L.S. Oliveira, R. Sabourin, F. Bortolozzi, C.Y. Suen, A methodology for feature selection using multiobjective genetic algorithms for handwritten digit string recognition, Int. J. Pattern Recognit. Artif. Intell. 17 (2003) 903–929, doi:10.1142/S021800140300271X.

[43] Z. Fei, B. Li, S. Yang, C. Xing, H. Chen, L. Hanzo, et al., A survey of multi-objective optimization in wireless sensor networks: metrics, algorithms and open problems, in: IEEE Commun. Surv. Tut., 19, 2016, pp. 1–38, doi:10.1109/COMST.2016.2610578.



Eduardo Viegas received the BS degree in computer science in 2013 and the MSC degree in computer science in 2016 from PUCPR. He is currently working towards his PhD degree in computer science at PUCPR. His research interests include machine learning and security.

Altair Olivo Santin received the BS degree in Computer Engineering from the PUCPR in 1992, the MSc degree from UTFPR in 1996, and the PhD degree from UFSC in 2004. He is a full professor of Graduate Program in Computer Science (PPGIa) and head of Security & Privacy Lab (SecPLab) at PUCPR. He is a member of the IEEE, ACM, and the Brazilian Computer Society.



Luiz S. Oliveira received his B.S. degree in Computer Science from UP, the M.Sc. from UTFPR, and Ph.D. degree in Computer Science from École de Technologie Supérieure, Université du Quebec in 1995, 1998 and 2003, respectively. From 2004 to 2009 he was professor of PUCPR. In 2009, he joined the UFPR, where he is professor of the Department of Informatics.