A Reliable and Energy-Efficient Classifier Combination Scheme for Intrusion Detection in Embedded Systems

Eduardo Viegas , Altair Santin , Luiz Oliveira , André França , Ricardo Jasinski , Volnei Pedroni

PII: S0167-4048(18)30617-5
DOI: 10.1016/j.cose.2018.05.014
Reference: COSE 1351

To appear in: *Computers & Security*

Received date: 8 October 2017
Revised date: 23 May 2018
Accepted date: 28 May 2018

Please cite this article as: Eduardo Viegas , Altair Santin , Luiz Oliveira , André França , Ricardo Jasinski , Volnei Pedroni , A Reliable and Energy-Efficient Classifier Combination Scheme for Intrusion Detection in Embedded Systems, *Computers & Security* (2018), doi: 10.1016/j.cose.2018.05.014

A Reliable and Energy-Efficient Classifier Combination Scheme for Intrusion Detection in Embedded Systems

Eduardo Viegas[1], Altair Santin[1], Luiz Oliveira[2], André França[3], Ricardo Jasinski[3], and Volnei Pedroni[3]

[1]Pontifical Catholic University of Parana, Curitiba, 80215-901, Brazil
{eduardo.viegas, santin}@ppgia.pucpr.br

[2]Federal University of Parana, Curitiba, 80060-000, Brazil
lesoliveira@inf.ufpr.br

[3]Federal Technological University of Parana, Curitiba, 80230-901, Brazil
andref@alunos.utfpr.edu.br, rjasinski@ieee.org, pedroni@utfpr.edu.br

Abstract—Embedded systems (electronic systems with a dedicated purpose that are part of larger devices) are increasing their relevance with the rise of the Internet of Things (IoT). Such systems are often resource constrained, battery powered, connected to the internet, and exposed to an increasing number of threats. An approach to detect such threats is through an anomaly-based intrusion detection with machine-learning techniques. However, most of these techniques were not created with energy efficiency in mind. This paper presents an anomaly-based method for network intrusion detection in embedded systems. The proposed method maintains the classifier reliability even when network traffic contents changes. The reliability is achieved through a new rejection mechanism and a combination of classifiers. The proposed approach is energy-efficient and well suited for hardware implementation. The experiments presented in this paper shows that the hardware versions of the machine learning algorithms consume 46% of the energy used by their software counterparts, and the feature extraction and packet capture modules consume 58% and 37% of their respective software counterparts.

Keywords—Classifier design and evaluation; Feature evaluation and selection; Machine learning; Energy-aware systems; Network-level security and protection; System-on-a-Chip; Field-Programmable Gate Array

1 INTRODUCTION

Most of the Internet of Things (IoT) and so-called "smart" devices are electronic devices containing embedded systems. Such systems are often battery-powered, resource constrained, and connected to the internet; therefore, they are exposed to an ever-increasing number of threats. According to OWASP [1], IoT devices are vulnerable to attacks and in general provide unsafe web interfaces and network services, raising security and privacy concerns.

An approach to detect attacks at the network level is through anomaly-based intrusion detection [2]. In general, this approach treats intrusion detection as a pattern recognition problem [3] that can be addressed with machine learning techniques. The use of machine learning makes it possible to detect variations of existing attacks and even new kinds of attack [3][25].

Machine learning uses an inference algorithm (a classifier) that learns the attack behavior in a training phase. In this phase, each input event occurring in the network is labeled as normal or attack (e.g., an intrusion attempt). The events are seen by the classifier as a set of attributes composing a feature vector. A set of feature vectors composes a dataset. The part of the dataset used to obtain the classifier model is called the training dataset.

A classifier's accuracy is estimated using a test set, in a process commonly referred to as model testing. Modeling and testing a classifier aimed at network intrusion detection is challenging because the network traffic content in real-world network environments is highly variable; moreover, it is infeasible to represent all possible network profiles in a training dataset.

An important requirement of an anomaly-based intrusion detector is its reliability. The higher the detection rate, the more accurate is the classifier. However, when it is not possible to obtain a detection rate close to 100%, the second most important goal in classifier development is to guarantee that when it associates a class to an event, this classification is reliable. If a security system operator cannot trust the alerts generated by an intrusion detection engine due to a high number of false positives or negatives, future alerts might be disregarded even though most of them are correct. For this reason, it is often preferable to have a classifier that is reliable rather than one with a higher detection rate but also a high rate of false positives or negatives.

Several approaches are proposed in the literature to improve a classifier's accuracy [4] [32]. A common solution is to combine the output of several classifiers, generally by voting techniques [5]. Although this approach usually improves the classifier accuracy, it may be inadequate for embedded systems, because the processing power and energy consumption increase with the number of used classifiers.

Another approach is to provide a classification probability along with the classification result and reject the classifier output when this probability is low [8]. This is a common approach in other fields such as bank checking systems [6], spam detection [7], and facial recognition [12]. However, even though rejection techniques have shown promising results in other fields and classifier combinations have been used in other areas [9][10][11],

rejection techniques are still not common in intrusion detection.

A rejection technique may significantly improve the classification reliability. When the network traffic content changes, the rejection rate could be increased to maintain the classification accuracy stable. Moreover, when the rejection increases, this may indicate that the detection models should be updated. However, if this update is not possible, the intrusion detection alerts will continue to be reliable. The events that are not classified are said to be rejected by the classifier.

A common approach to detect changes in the network traffic is to analyze the history of extracted features values [13] in a fixed- or variable-length time window [14] [41]. A new network profile is detected when the features values are significantly different within this window [15]. A drawback of this approach is that it is computationally expensive, because it requires storing and analyzing a large number of events [16] to detect network changes. Therefore, this approach may be infeasible in resource-constrained devices.

To reduce the energy consumption and increase the system throughput [17], network security algorithms can be implemented in hardware [18][19]. The algorithms are usually coded using a hardware description language (HDL) such as VHDL (Very High Speed Integrated Circuits HDL), and the circuits are implemented in hardware using a System-on-a-Chip (SoC) or Field-Programmable Gate Array (FPGA) [20]. Such approaches take advantage of the parallel processing capacity inherent to a hardware implementation, allowing the use of several classifiers, often with lower energy consumption [17]. However, the implementation of security mechanisms in hardware has several drawbacks. New attacks are discovered daily; to be usable in real-world environments, the classifiers must be able to cope with new attack profiles and still provide a reliable classification. If the intrusion detect engine is implemented on an FPGA, unless partial reconfiguration is used, an update of the detection engine or classifier model may require reprogramming the entire chip [20].

This paper presents a new method using rejection techniques to improve the classification reliability in an anomaly-based intrusion detection engine. We present and evaluate a hardware implementation that is suitable for resource-constrained embedded systems. In summary, our main contributions and novelties are:

- A new rejection method suitable for embedded systems, providing classification reliability even when the network traffic behavior changes.
- A new method to define detection rates for machine-learning-based intrusion detection algorithms. The expected anomaly-based detection properties are used in combination (detection of known, similar and new attacks), allowing the obtainment of the best parameter vector arrangement for each situation.
- A new, hardware-friendly implementation of a reliable anomaly-based intrusion detector using the proposed rejection technique. Our implementation acquires network packets directly from the network and performs reliable classification even in the presence of network traffic changes, using multiple classifiers. The hardware version has a significantly lower energy consumption than the equivalent software implementation, making it suitable for use in embedded systems.

The remainder of this paper is organized as follows. Section 2 presents the background work and the main challenges associated with anomaly-based intrusion detection. Section 3 explains in detail the proposed anomaly-based detection method and the testbed environment. Section 4 presents the evaluation of the proposed rejection methods. Section 5 describes the software and hardware implementations. Section 6 presents the energy consumption measurements of the software and hardware implementations. Section 7 presents related works. Finally, Section 8 summarizes the conclusions and the main contributions of this work.

2 BACKGROUND

A Network-based Intrusion Detection System (NIDS) captures and classifies events (network packets, flows, or connections) in a network environment. A typical anomaly-based workflow for NIDS is shown in Fig.1. Initially, a Packet Capture module collects and filters the network traffic. Next, the filtered events (e.g., network packet headers or connection flow statuses) are sent to the Feature Extraction module, which selectively obtains a set of features (attributes relevant to the purpose of attack characterization) and assembles a feature vector. Finally, the Detection Engine assigns a class (normal or attack) to the event.

Several techniques are used in the literature to assign a class to an event; pattern recognition is the most common approach [2]. This approach uses an inference algorithm to build a model by learning the event classes from a training dataset - a file containing a set of previously labeled features vectors. After the classifier has been

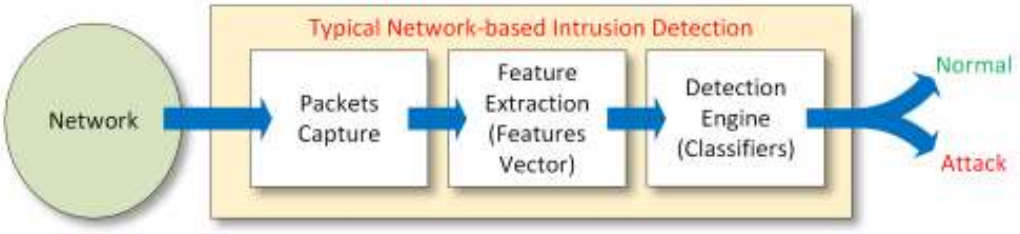

Fig. 1. Typical processing flow for an anomaly-based NIDS

trained, the event class can be predicted using the learned attack model. Events from different classes but with similar feature values can be wrongly classified by the classifier, resulting in false positive or negative detections (FPND).

Usually, the model development process uses three distinct datasets: one for training, one for validating, and one for testing the model. The training dataset is used to obtain the attack model, whereas the validation dataset can be used to fine-tune the model parameters. Due to FPND, during the model obtainment process an accuracy rate is estimated, using the test dataset. If necessary, an iterative process involving the training, validating and testing datasets can be performed to improve the model accuracy rate.

As reported in the literature, the combination of classifiers can improve a system's accuracy rate [5] [21]. Most classifier combination schemes use majority voting to assign a final class to an event.

2.1 Anomaly-based Network Intrusion Detection

An important feature of anomaly-based intrusion detection is its ability to detect new kinds of attack using the same model [3][25]. This is particularly important because network traffic and attack profiles are frequently changing, either by the adoption of new technologies (services) or by the emergence of new kinds of attacks. However, despite its extensive presence in the literature [2][27][28], anomaly-based intrusion detection is not commonly used in commercial products, mainly because it faces several challenges when compared with other approaches such as signature-based techniques [25][26].

The constant changes in network traffic make it difficult to create representative datasets [25]. Training a machine learning algorithm requires a significant number of samples from each class – in this case, a large number of packets representing both normal and attack network traffic [26].

Changes in network traffic may require periodic updates of the intrusion detection model. Due to the lack of public intrusion databases reflecting such characteristics, many works found in the literature simply use network traffic that does not change over time [25]. Such works assume that the detection accuracy rate obtained during the model training, validating and testing will remain valid for a long period in real-world environments. However, in practice, when the traffic content changes, the model accuracy probably changes as well, and the classifier may become less effective.

Unlike other areas in which misclassifications are acceptable, intrusion detection systems pay a high cost for FPND [25]. An intrusion attempt misclassified as a legitimate access may compromise an entire system. Therefore, to be reliable, a NIDS must be able to deal with network traffic changes and still provide a reasonable detection rate.

3 Event Rejection Method

The design of a security mechanism in hardware is not a trivial task. Besides the difficulties of adapting algorithms that were initially developed with the flexibility of software in mind, model updates may be difficult due to the more static nature of a hardware implementation. The approach described in this paper uses a rejection technique and a combination of classifiers to provide a more reliable detection. This solution aims at being hardware-friendly, energy-efficient, and reliable over extended periods, even though it does not classify all the input events.

3.1 Changes in Feature Values Distribution

When a classifier is operating, its accuracy depends on the feature values distribution being similar to that of the training dataset (usually composed of real network traffic). If the distribution changes significantly, the classifier model should be updated, or its accuracy may decrease. This update usually requires expert knowledge to label new events and to rebuild the model, which may not be practical in real-world environments. To test a classifier designed to operate in such environments, we need a method to assess whether it is still reliable even when the network traffic changes. Here we describe an evaluation scenario and propose an event rejection method that allows the classifier to operate reliably even when it cannot be easily updated.

To overcome the limitations of other works in the literature, we propose a rejection method that takes into account the frequent content changes observed in real-world network traffic. We also propose the usage of several independent classifiers using different machine-learning algorithms. After each classification, we check whether there are enough similarities between the classifier outputs class (normal or attack) and the class occurrence observed in the training dataset. If there is not a predominant match, the classification is deemed unreliable and the event should be rejected because the features used to build the model and the current event are not similar enough for a reliable classification. An event rejection means that none of the classifiers can reliably assign a class to an input event; in this case, the event is rejected rather than being potentially incorrectly classified.

3.1.1 Scenario

Fig. 2 shows a real-world scenario whose feature distribution changes over time. It considers the feature set of SYNFlood attacks [29] as baseline in the attack model. If an HTTPFlood attack [29] occurs, it can still be detected because the feature distribution of the two attacks are similar. However, if the network traffic changes significantly (as in an Exploit attack [29]), the classification output becomes unreliable due to the significant change in the feature set. In such cases, if the model cannot be updated, another technique should be used to provide a reliable classification.

3.1.2 Rejection Engine

One way to detect changes in the network traffic profile is to monitor the distribution of values in the extracted feature set (Fig. 2, Exploit Attack). A significant change in feature distribution may indicate that a new attack is

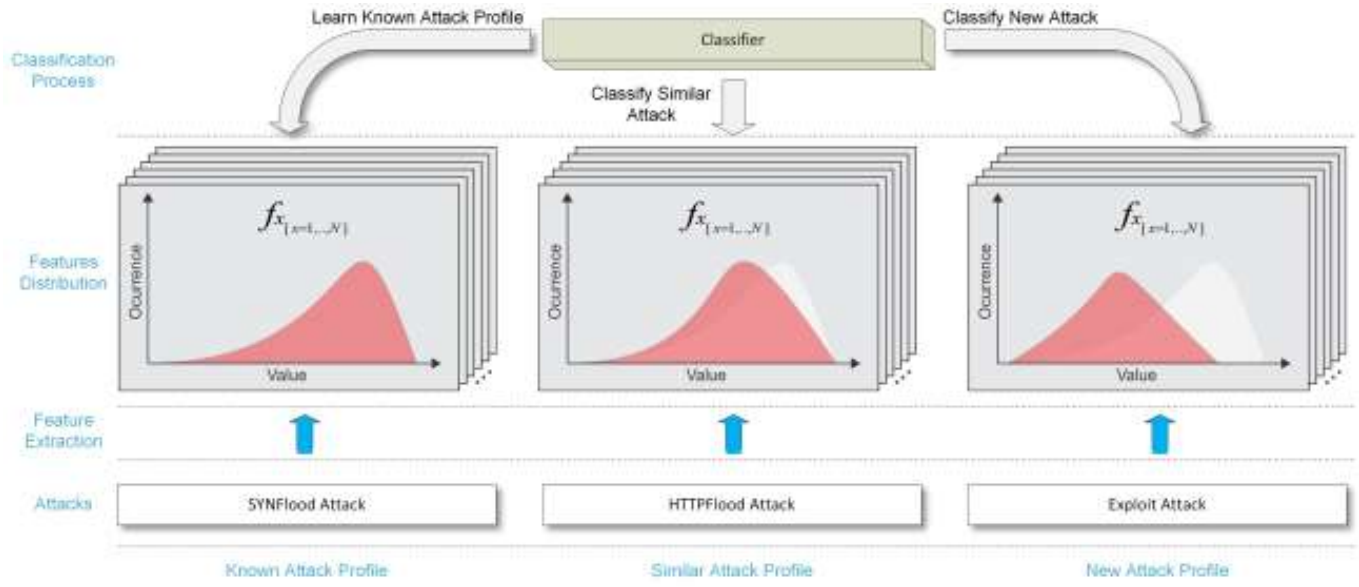

Fig 2. Changes on features distribution, considering SYNFlood Attack as reference.

occurring. However, it is not easy to detect profile similarities from network events occurring in real time.

As shown in [18][30][31][47], in a NIDS, approximately 50 features must be taken into account for anomaly-based intrusion detection. If each feature is represented with 2 bytes [18], a single event will require 100 bytes for storage. Considering a feature distribution history of 100 events, 10 KB are required in order to store the data for detecting changes in the distribution. The amount of memory storage, processing power and energy required to update the feature distributions and to detect changes could be impractical for embedded systems. To overcome these issues, we defined two ranges for each attribute (one for each class) to determine whether a feature value is valid. When an extracted feature lies within the appropriate range, the feature is considered valid.

To evaluate our method, we used three traffic scenarios: a baseline scenario, a scenario with network traffic changes but similar to the baseline scenario, and a scenario with new attacks (Fig. 2). The baseline scenario was used to obtain the rejection range thresholds and the attack models; the other scenarios were used to evaluate the rejection method.

For each feature (fx, $x=1,2,\ldots,N$) and class (normal or attack), two rejection limits or thresholds (t_{lower} and t_{upper}) were computed. The limits define the range within which a feature value is valid. The range is class-specific because the feature distribution for each class is different. The thresholds are defined with respect to an $\propto$ value (Fig. 3), which establishes a percentage of instances in the validation dataset that fall outside the defined thresholds, but still provide the desired model reliability. To determine the value of $\propto$, an experimental analysis must be performed (Section 4.2).

For each feature fx, $profile_{similarity\_fx}$ = 1 if the values for fx lie in the threshold interval ($t_{interval}$): t_{lower} < value (f_x) < t_{upper}; otherwise, $profile_{similarity\_fx}$ = 0. For example, $profile_{similarity\_fx}$ = 1 for the attack profile (Fig. 3) and $profile_{similarity\_fx}$ = 0 in Fig. 4 for both profiles.

If N denotes the number of features in the feature set, the profile instance similarity ($profile_{similarity}$) is defined according to (1):

$$profile_{similarity} = \frac{\sum_{x=0}^{N} profile_{similarity\_fx}}{N} \quad (1)$$

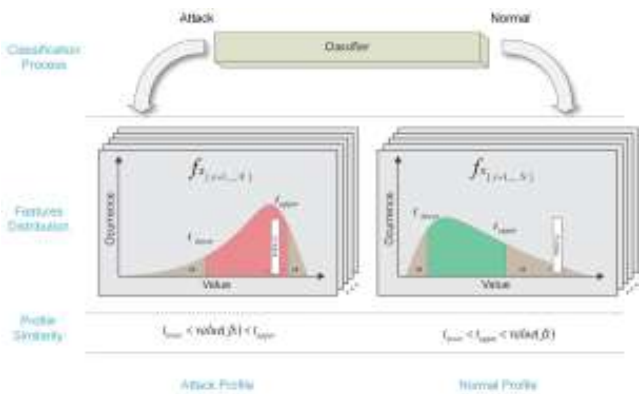

Fig 3. Features within the threshold range for a class (attack).

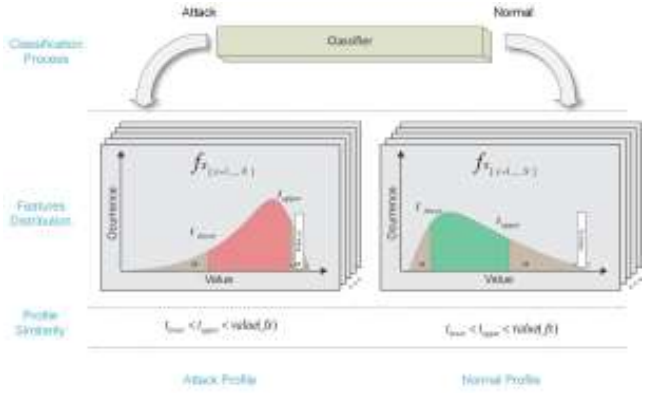

Fig 4. Feature outside the threshold range for both classes.

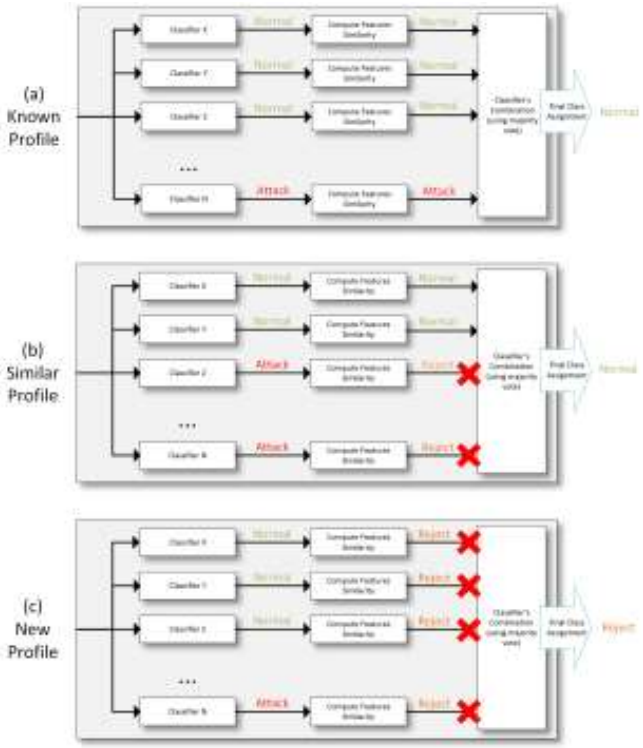

Fig 5. Final class assignement using majority vote as classifier combination

A classifier output should be rejected when it presents a low $profile_{similarity}$ (e.g., $profile_{similarity} < 0.7$); otherwise, the event is labeled with the class informed by the model (Fig. 5a and 5b). Using this approach, we are able to establish the profile similarity without the need to keep the feature values history to identify a change in feature distribution.

The output of the combined classifier is assigned via a combination algorithm (Fig. 5), choosing the majority of the outputs of the individual classifiers whose outputs were not rejected. In the example of Fig. 5c, the output is rejected because no individual classifier output is valid, while in Fig. 5a and Fig. 5b the output is accepted because there is at least one classifier that can reliably classify the event.

3.2 Changes in the Distribution of Feature Value

Although there are some public datasets and proposed validation approaches in the literature [33][34][35][36][37], most lack desirable IDS properties such as reproducibility, update capacity, real and valid traffic content, correct class labeling, and fully-compliant attack implementations. Therefore, to validate the proposed rejection method, it was necessary to develop a testbed environment to ensure it had the desired IDS properties (Fig. 6).

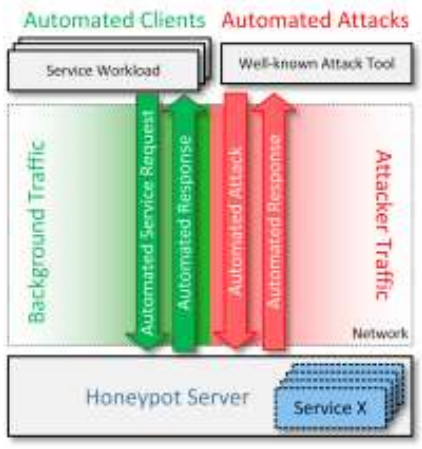

Fig. 6.Network traffic generation process.

Differently from most works in the literature [35][36][37], the traffic content in our scenarios was not recorded from real-world network traffic, thus avoiding privacy issues such as sensitive data exposure [26]. We used a controlled environment with automatic traffic generation mimicking a real-world environment. This is the same approach used in our previous work [18], but with a larger number of scenarios. The background traffic (normal or legitimate events) is generated with workload tools (Fig. 6, Background Traffic), whereas the attack traffic is generated with well-known, standard exploit tools (Fig. 6, Attacker Traffic). All requests are performed to a honeypot server – a highly interactive virtual server that implements vulnerable servers on a network, normally used to study attacker behavior.

The next sections briefly detail the method used to create the intrusion datasets, the conceived scenarios, and the dataset properties used in our work.

3.2.1 Background Traffic Creation Method

In an intrusion dataset, the background traffic must contain the normal and expected network activities. In a real-world environment, the normal traffic content of a client is service-dependent (e.g., a client browsing a website behaves differently from a client sending an e-mail).

Our approach is to use virtual machines running real software applications performing real network requests and responses at pseudo-random intervals. The background traffic contains network packets generated by the clients and server. We use a honeypot tool as server for the client requests; the server receives the requests, interprets them, and provides proper replies, generating the background traffic (Fig. 6, Background Traffic). In this way, we can provide real responses to the clients in a controlled environment.

To generate the client-side traffic, we use a specific workload tool for each service. The generated traffic consists of requests that could be observed in a real-world network; they are all valid, well-formed packets, including both request and response messages. The behavior of each virtual client was defined to request one type of service with pseudo-random contents. Each client requests its predefined service indefinitely, with different contents and different time intervals between the requests.

3.2.2 Attack Traffic Creation Method

To create the attack traffic, we used the same method proposed in [18], using well-known and *de facto* standard tools to generate the attacks (Fig. 6, Attacker Traffic). In this way, we ensure that the attacks are correctly implemented and reproducible.

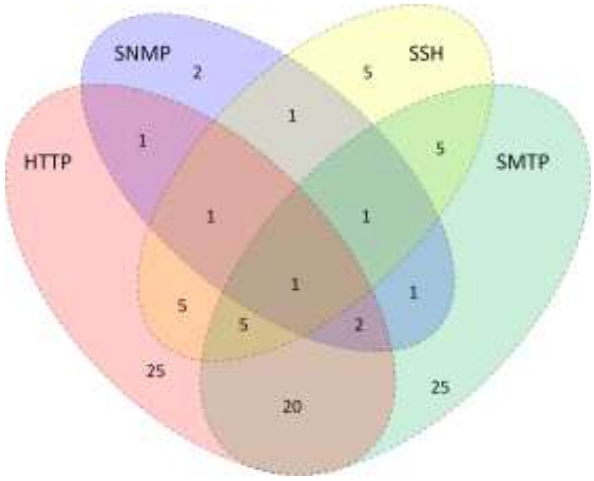

Fig 7. Client distribution for each service in the deployed scenarios.

the attacks, the simulation setup is composed of three distinct scenarios, with a duration of 30 minutes each. All the traffic is logged so that the scenario can be replayed later if needed.

Each scenario consists of 100 interconnected clients that request one or more services to a single server. The number of attackers varies according to the scenario (Table 2), while the number of clients requesting each service is shown in the Venn diagram of Fig. 7. The honeypot server and the clients generating the normal background traffic use the Ubuntu 14.04 OS. The attacker machines run Kali Linux 1.0.0a. The honeypot server runs honeyd 1.5c.

TABLE 2
ATTACK AND TOOLS USED IN BENCHMARK SCENARIOS

Scenario	Attack	Tool	Description
Baseline (model-known attacks)	UDPScan	Nmap	Searches for open UDP ports varying the attack frequency and duration.
	SYNScan	Nmap	Searches for open TCP ports by sending TCP packets with the SYN flag set while varying the attack frequency and duration.
	NULLScan	Nmap	Searches for open TCP ports by sending TCP packets without flags set while varying the attack frequency and duration.
	TCPConnect	Nmap	Searches for open TCP ports by completing the three-way handshake while varying the attack frequency and duration.
	FINScan	Nmap	Searches for open TCP ports by sending TCP packets with the FIN flag set while varying the attack frequency and duration.
	XMASScan	Nmap	Searches for open TCP ports by sending TCP packets with the FIN, PSH and URG flags set while varying the attack frequency and duration.
	ACKScan	Nmap	Searches for open TCP ports by sending TCP packets with the ACK flag set while varying the attack frequency and duration.
Similar attacks	OS Fingerprint	Nmap	Identifies the OS of the target (https://nmap.org/book/osdetect.html) while varying the attack frequency and duration.
	Service Fingerprint	Nmap	Identifies the target's services and their versions (https://nmap.org/book/man-version-detection.html) while varying the attack frequency and duration.
New attacks	Vulnerability Scan	Nessus	Identifies service level vulnerabilities while varying the attack frequency and duration.

3.3 Background Traffic Creation Process

The following services were represented in our testbed environment: HTTP (Hypertext Transfer Protocol), SNMP (Simple Network Management Protocol), SMTP (Simple Mail Transfer Protocol), NTP (Network Time Protocol), and SSH (Secure Shell). DNS (Domain Name System) requests were also generated as a consequence of using the listed protocols. These services were selected because they reflect the most used network services nowadays, as specified in [38].

To create the honeypot server (Fig. 6), we used the honeyd [39] tool. To implement the clients, we used a workload tool automated with custom scripts. All tools are available at secplab.ppgia.pucpr.br/eeids.

To provide traffic variability, each client randomly varies the requested content according to the service description shown in Table 1. The time between requests varies between zero and four seconds. This method can approximately mimic the behavior of a user browsing a webpage and sending e-mails, for instance.

To analyze the capability of the classifiers to detect

TABLE 1
SERVICES USED FOR BACKGROUND TRAFFIC GENERATION

Service	Description
HTTP	1,000 most visited worldwide websites were downloaded (www.alexa.com/topsites) and hosted on the honeypot; each HTTP client requests a pseudorandom website from such set of content.
SMTP	Each SMTP client sends a mail with a 50-400 bytes subject and 100-4.000 bytes in the body.
SSH	Each SSH client logs into the honeypot host and executes a random command from a list of 100 possible commands.
SNMP	Each SNMP client walks through a predefined MIB from a predefined list of possible MIBs.
DNS	Every name resolution is also made to the honeypot server.

TABLE 4
ACCURACY FOR EACH SCENARIO USING THE OBTAINED CLASSIFIERS

Classifier / Accuracy	Scenario		
	Baseline	Similar	New attack
DT	99.97 %	98.62 %	64.66 %
NB	99.75 %	99.23 %	57.38 %
LDA	99.44 %	98.39 %	56.00 %
Combination	99.75 %	99.14 %	70.29 %

A single LAN network at 100 Mbps connects all hosts.

TABLE 3
GENERATED TRAFFIC FOR EACH SCENARIO

Scenario	Traffic (Packets)			Database Size (Megabytes)
	Background	Attack	Total	
Baseline	28,618,365	36,628	28,654,993	8,476
Similar	28,477,884	10,441	28,448,325	8,499
New	28,391,914	17,753	28,409,667	8,512

The network speed allowed us to capture the generated traffic on a single host without dropping any packets [40]. Both normal requests and attacks are generated targeting the honeypot server (Fig. 6). The generated traffic is logged on the honeypot server itself.

3.4 Rejection Method Evaluation

To mimic a real-world environment, we considered three distinct scenarios (Fig. 2) representing a traffic profile that changes over time: (i) a known attack profile (baseline dataset), (ii) a similar attack profile (similar dataset), and (iii) a new attack profile (new dataset).

The baseline dataset is composed of normal background traffic and a known set of attacks (Table 2). This dataset is used to train the model and to establish the reference accuracy rate for the classifiers.

After the attack model has been obtained, we use the other scenarios to evaluate the classifier's ability to detect similar and different attacks that were not considered in the model training phase. Only the attack traffic changes across the scenarios; the normal traffic remains the same.

The baseline scenario contains probing attacks, a kind of attack in which an opponent gathers information about a host in the network (such as open ports, service versions, and operating system fingerprints). In this baseline scenario, the attacks occur mainly at the network protocol level.

The second attack scenario also consists of probing attacks; however, they aim at gathering application-level information. This kind of attack has a lower traffic content variation compared with the baseline dataset.

Finally, the third attack scenario represents the occurrence of a new kind of attack, using exploit attacks. This dataset represents real-world environments where new attacks (not previously learned by the model) are discovered over time. Table 2 presents the tools used to generate the attacks.

To generate the databases, each scenario was deployed for 30 minutes. The number of clients requesting each kind of service is depicted in Fig. 7. The behavior of each client is described in Table 1 and the attack profiles are shown in Table 2. The traffic amount for each scenario is shown in Table 3.

4 EVENT REJECTION EVALUATION

The datasets described in section 3.4 and the evaluation method presented in Fig. 2 were used to evaluate the proposed rejection method. Three energy-efficient classifiers [17] were used during the evaluation: Decision Tree (DT), Naïve Bayes (NB), and Linear Discriminant Analysis (LDA). The following sections describe how the classifier models were built and how the proposed rejection method was evaluated.

4.1 Model Obtainment Process

In our experiments, we have used two distinct detection approaches. First, a single classifier using the DT, NB or LDA algorithm; second, a combination of the three classifiers using majority voting, as explained in section 3.1.2 (Fig. 4).

TABLE 5
FEATURES GROUP

Group	No. of Attributes	Description	Example
Header-based	30	Features extracted directly from the packet header	SYN flag from TCP protocol
Service-based	17	Features regarding the communication between two hosts on the network	Bytes sent in the last two seconds
Header-based	7	Features regarding the communication between two services on the network	TCP connection status

The naturally occurring amounts of packets corresponding to normal and attack events are distinct, because most of the logged events are normal. However, during the dataset generation, we have selected the same number of events from each class. This allowed us to compute the classifier accuracy without the need for verification of false-positives and false-negatives during the model generation.

Our stratification process consisted of randomly selecting 25% of the events from the least frequent class, and then selecting the same number of events from the other class. The datasets were obtained using a stratification process with 25% for training, 25% for validation, and the remaining events (50%) for testing. The test dataset is composed of events that were not used to obtain any classifier model.

To obtain the classifier models, we used the Weka [48] framework version 3.7.12. For the NB classifier, all the numerical attributes were discretized according to the method proposed in [42]. The C4.5 decision tree algorithm was used with a confidence factor of 0.25. The Fisher's method [43] was used for the LDA classifier.

After the models were obtained using the baseline dataset, they were tested in the three available scenarios. Each classifier had its average accuracy measured, as defined by the average of true-positive and true-negative rates. Table 4 presents the resulting accuracy in each scenario, for each single classifier in the combination

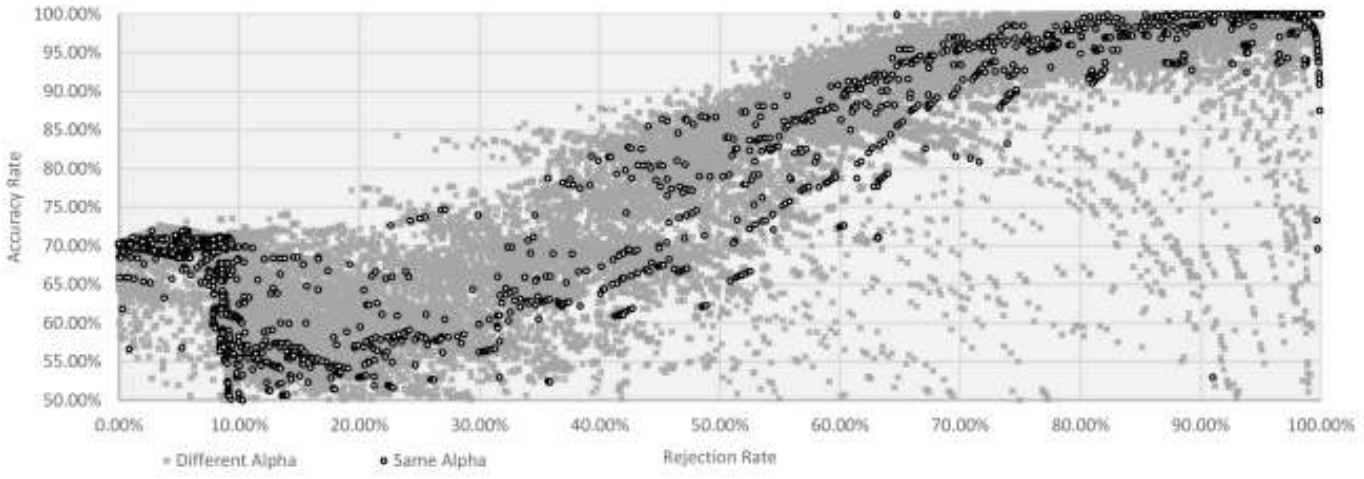

Fig 8. Accuracy-rejection tradeoff for the combination technique while detecting new attacks.

scheme and also for the combined classifier, using the testing dataset and without the proposed rejection scheme.

All classifiers presented a reasonably good performance when used in the baseline testing dataset. The best accuracy rate, 99.97% was obtained with the DT classifier. The worst classifier accuracy rate, 99.44%, was achieved by LDA, although it was only 0.53% lower than DT. When evaluated with the similar scenario, the classifiers were able to detect events with an average accuracy drop of 0.88% compared to the baseline scenario. For new attacks, however, the accuracy decreased on average by 37.64%. The best accuracy rate when detecting new attacks was obtained with the combination/voting classifier, with an accuracy of 70.29% (Table 4). None of the used classifiers or methods were able to maintain their baseline accuracy when detecting similar or new attacks. If such classifiers were used in a real-world environment, their accuracy would likely drop over time due to changes in the traffic or attack profiles.

4.2 Evaluation of the Proposed Rejection Method

Our proposed rejection method aims at maintaining the classifier reliability over time even in the absence of model updates. To achieve this goal, our class assignment (Fig. 4) must reject potentially wrong classifications. Therefore, the evaluation tests aim at checking the detection accuracy, while still rejecting as few events as possible.

TABLE 6
ACCURACY-REJECTION TRADEOFF FOR EACH DATASET USING THE POINTS MARKED IN FIG. 10 (SCENARIO).

Rejection Rate	Classifier (Normal Alpha, Attack Alpha)	Dataset/Scenario					
		Known attack		Similar attack		New attack	
		Acc. (%)	Rej. (%)	Acc. (%)	Rej. (%)	Acc. (%)	Rej. (%)
No Rejection	DT (n.a., n.a.)	99.97	-	98.62	-	64.66	-
	NB (n.a., n.a.)	99.75	-	99.23	-	57.38	-
	LDA (n.a., n.a.)	99.44	-	98.39	-	56.00	-
	Combination (n.a., n.a.)	99.75	-	99.14	-	70.29	-
Low Rejection	DT (0.55, 0.18)	99.97	0.21	98.70	1.05	**67.37**	4.82
	NB (0.62, 0.27)	99.75	0.14	99.23	0.25	59.66	4.07
	LDA (0.64, 0.29)	99.44	0.14	98.48	0.31	58.15	3.98
	Combination (0.53, 0.16)	99.75	0.01	99.14	0.00	**72.66**	3.31
Average Rejection	DT (0.83, 0.37)	99.98	5.40	98.80	5.77	**74.90**	24.14
	NB (0.88, 0.51)	99.75	5.41	99.21	5.88	67.94	23.50
	LDA (0.88, 0.51)	99.65	5.71	99.65	6.00	66.37	24.99
	Combination (0.81, 0.25)	99.76	0.25	99.14	0.15	**84.27**	23.16
High Rejection	DT (0.90, 0.51)	99.99	37.27	99.87	25.79	**99.92**	59.57
	NB (0.90, 0.51)	99.96	37.34	99.92	25.42	99.92	59.61
	LDA (0.90, 0.51)	100.00	37.38	99.87	25.57	99.86	59.62
	Combination (0.90, 0.51)	99.95	37.27	99.92	25.38	**99.86**	59.52

Due to the number of used features (54 features, Table 5), evaluating all possible values of $\propto$ for each feature and profile similarity (section 3.1.2, Fig. 3) was infeasible. Therefore, we have performed two tests for each final class assignment (Fig. 4). The first test, named Different Alpha, used different $\propto$ values (Fig. 3) for each feature group (header-based, service-based, and host-based, Table 5), whereas the second test (named Same Alpha) used the same $\propto$ for all features. The profile similarity varied from 0% to 100% in 1% increments. The rejection rate is the ratio between the number of rejected instances

and the total number of instances in the test set. The accuracy rate *vs* rejection rate tradeoff using the combined classifiers for the detection of new attacks is shown in Fig. 8.

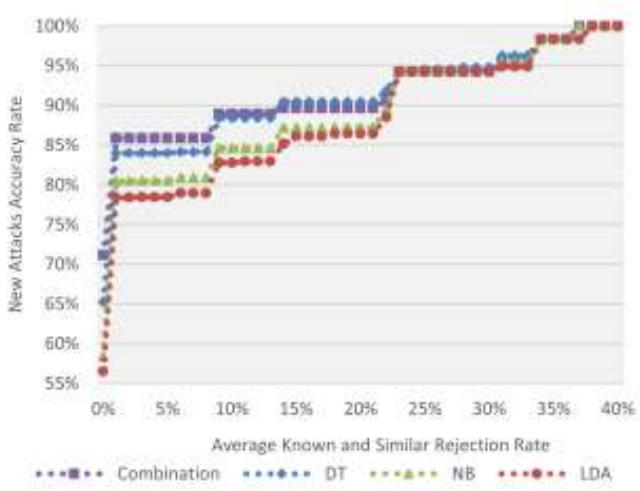

Fig. 9. Tradeoff between the accuracy improvement for new attacks and the rejection of known and similiar attacks.

It is possible to note that in most cases there is a direct relation between accuracy and rejection, regardless of the ∝ technique (Fig. 8, Different Alpha and Same Alpha). One may notice that different distributions of feature values allowed improving the accuracy rate while rejecting fewer instances. The classifier combination scheme was able to reach an accuracy rate of 100% while rejecting 59.52% of instances in the new attack dataset.

All evaluated classifiers were able to reach an accuracy of 100% at the cost of a 60% rejection rate for the new attack dataset (attacks not previously known to the model). The combination classifier provided the best accuracy with a minimum rejection rate considering all scenarios, outperforming the best single classifier (DT) by about 5% in the low rejection rate setting, and by about 10% in the average rejection setting (Table 6).

In the real world, it is not possible to choose a different set of thresholds for each event, because the classifier is unable to determine whether an event is a known attack, a similar attack, or a new one. Therefore, the choice of a set of thresholds must be made taking into account the tradeoff between accuracy and rejection rate. Fig. 9 shows the accuracy-reject tradeoff between the accuracy in detecting new attacks and the rejection rate for known and similar attacks, using the same set of thresholds during the detection. The graph shows that it is possible to maintain the accuracy for the detection of new attacks, but at the cost of an increased rejection rate for known and similar attacks. For instance, it is possible to maintain the accuracy rate at 95% in a scenario with new attacks, at the cost of rejecting 31% in average of the events in the other two scenarios (known and similar attacks).

The set of thresholds should be established according to the user goals. If certain lenience for accuracy is acceptable, fewer events will be rejected, but the class assignment will be more susceptible to errors. Table 6 shows the accuracy-reject relationship for each dataset, for the rejection settings highlighted in Fig. 10, using the same set of thresholds. Four rejection rate settings (no rejection, low rejection, average rejection, and high rejection) were selected for the new attacks dataset. The same

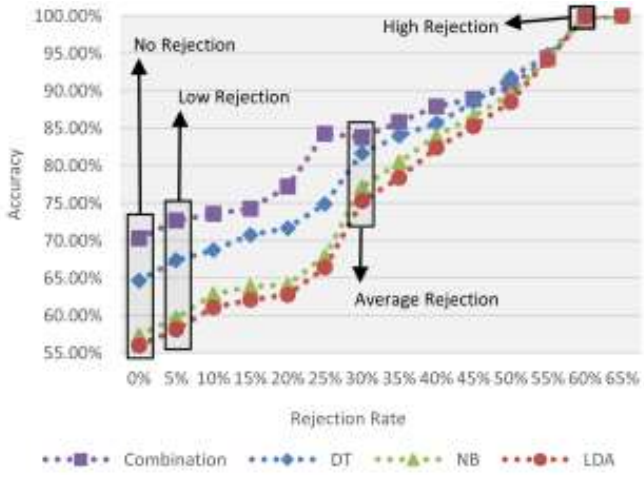

Fig. 10. Tradeoff between Accuracy and rejection rate, for each classifier in new attacks dataset.

set of thresholds were used in the other scenarios to investigate the rejection rate impact for the known and similar attack datasets. The obtained results are shown in Table 6.

The Average Rejection setting presented the best accuracy-reject tradeoff. The rejection method was able to improve the classification accuracy by 13.98% for new attacks while rejecting only 0.25% known attacks and 0.15% similar attacks, using the combination classifier (Average Rejection, Table 6). The combination classifier produced, in average, the best results when compared to the single classifiers at the same rejection rate interval. In summary, the proposed rejection method allowed the detection of new attacks while maintaining the classifier's overall reliability.

4.3 Comparison with other Rejection Approaches

Finally, two commonly used rejection approaches that rely on class probabilities, the Chow's rule [23] and the Class-related Reject Threshold (CRT) [24], were compared to our proposed method. Chow's rule defines a single rejection threshold for all classes, whereas CRT uses a different threshold for each class. For evaluation purposes, the combination classifier was used because it presented the best results (Table 6). The three approaches – CRT, Chow and the proposed approach – were evaluated using the New Attacks dataset. We used rejection rates from 0% to 100%. Fig. 11 shows the accuracy-reject tradeoff comparison for the evaluated approaches.

The proposed approach outperformed both existing techniques, CRT and Chow's rule. The traditional rejection approaches were not able to identify behavior changes and increased the classification confusion; the assigned class probabilities were high even for misclassified instances. In contrast, our approach was able to operate with fewer misclassifications in the presence of traffic behavior changes, reaching 100% accuracy while

rejecting 60% of the events.

4.4 Discussion of the Proposed Rejection Method

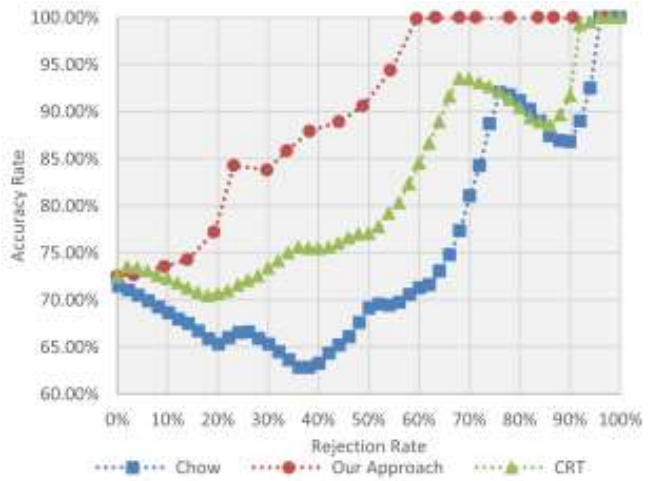

Fig. 11. Accuracy-rejection tradeoff, for the combination classifier in the new attacks dataset, using the evaluated rejection techniques.

The conceived scenarios reproduce three common situations in real-world intrusion detection systems. The Known scenario represents the environment behavior when the system was conceived, including all possible behavior variations that were known a priori. The Similar scenario represents the occurrence of attacks similar to those already known by the classifier. Finally, the New scenario represents the environment after an extended period of time, when new attacks (which were not publicly known when the model was trained) are created.

We presented an evaluation method that takes into account such situations. All tested detection schemes were able to detect known and similar behaviors with a reasonably high detection rate (an average accuracy of 99.72% for known behaviors and 98.84% for similar behaviors). However, when the system faces previously unseen attacks, its accuracy drops significantly (to an average of 62.08%). To ensure the classification reliability, we proposed the use of simple lower and upper thresholds, obtained from the feature distributions observed during the model training phase.

Due to the large number of used features, we have analyzed the contribution of distinct features according to their feature group. Our proposed embedded-friendly rejection method was able to guarantee the system reliability with a low accuracy-reject tradeoff, improving the accuracy in 13.98% for new attacks while rejecting only 0.25% of known behaviors using a classifier combination scheme.

Our experiments revealed other factors that should be taken into account by the anomaly-based intrusion detection community. A machine-learning classifier works by identifying similar behaviors and must have representative instances from all of the considered classes [25]. The assumption that a classifier will be able to detect new attacks only holds when their behavior is similar to the one used during the classifier training.

A machine learning detection system becomes unreliable when the traffic behavior changes. In our work, we have dealt with such issue by rejecting potentially non-reliable classifier decisions. However, in the literature, such effect is often ignored [26]. Network traffic is considered static, and the common machine learning evaluation schemes are adopted without taking into its dynamic characteristics.

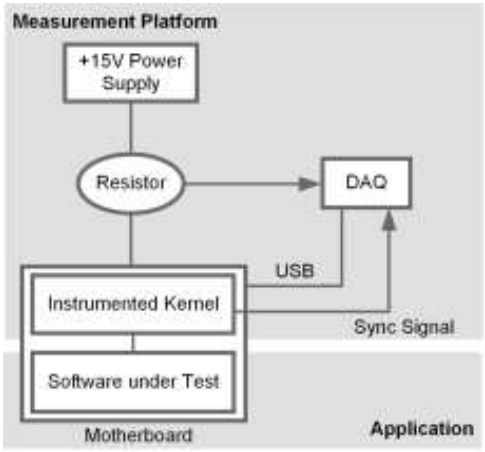

Fig. 12. Power measurement platform with application under

5 Classifiers Implementation

To evaluate the energy consumption of our classifiers and the proposed rejection scheme, we have implemented the anomaly-based intrusion detection schemes in both software (SW) and hardware (HW). The next sections describe the implementation of each platform.

To properly measure the energy consumption of our SW algorithms, we developed an energy measurement platform described in [18] and shown in Fig. 12. It allows isolating the energy consumed by the measured application from the consumption of other tasks running in the CPU. The platform is composed of a hardware environment, a measurement application, and an instrumented kernel (using a kernel-level probe module, or KPM). The KPM detects when the monitored software is running and triggers a signal in the motherboard's parallel port; while this signal is asserted, we know that the monitored software is running. We used a DN2800MT Atom motherboard with an N2800 CPU, 4 GB DDR3 RAM, and a 500 GB hard drive.

5.1 Software

The SW version of the proposed intrusion detection scheme is implemented as a sequence of four modules: Packet Capture, Feature Extractor, Classifier, and Rejection Logic. The output of each module is used as an input by the next one.

The Packet Capture module acquires network packets from the network interface and filters them. The network packets are collected using the libpcap library [44]. In our experiments, we capture all TCP, UDP and ICMP network packets.

The filtered network packets are sent to the Feature Extractor module, which extracts 54 features from the packet headers (Table 5). Service-based and host-based features are extracted using a hash table for performance

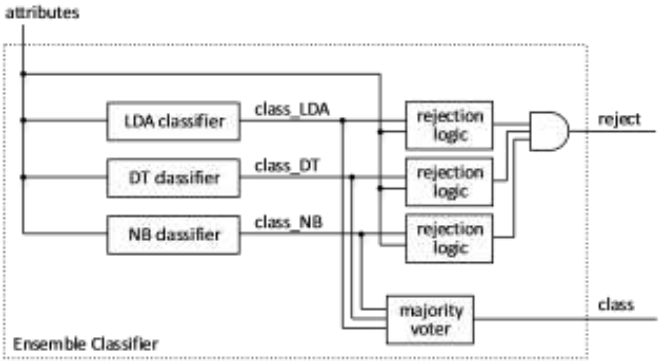

Fig. 13. Block diagram of the HW implementation of the combination classifier.

reasons. The network packet fields are used as keys for the hash function, and the hash value is used as an index into a table of cumulative feature values. This allows the feature history calculation to be performed as a cyclic, limited size list. The feature extractor implementation is described in more detail in [18]. The feature values are assembled into a feature vector and sent to the Classifier module, which classifies the network packet as normal or attack.

Finally, the Rejection Logic module computes the feature similarities for each decision and rejects them or not. The class assigned to the packet is the class with most occurrences among the accepted classifier outputs.

The proposed scheme was implemented in C++. To acquire the network packets, libpcap version 1.3.0 was used. The classifiers in the detection engine are a direct implementation of the classifiers provided by the Weka framework (Section 4.1).

5.1 Hardware

The hardware version of the proposed intrusion detection scheme was implemented as a system-on-a-chip (SoC) in an FPGA (Field Programmable Gate Array) device. The hardware that defines the system operation is implemented inside the FPGA; the only external component is an Ethernet PHY chip. The main processor is a 32-bit Altera Nios II soft-core CPU. The CPU uses both custom and ready-made peripherals and an internal bus to process the incoming data. The program and data memories are implemented using on-chip memory – dedicated memory blocks available in the FPGA. The hardware was designed in VHDL and synthesized in Quartus 13.0SP1 for an Altera Cyclone IV FPGA (EP4CGX150N).

The network system is able to operate at gigabit speed. The packets are received on the Ethernet PHY chip and processed in the Ethernet MAC module. The frames are stored in the on-chip memory via DMA.

The CPU firmware was coded in C and is responsible for five main tasks: configuring the receive DMA, configuring and managing the Ethernet MAC core, allocating packet descriptors in the on-chip memory, coordinating the bus data transfers between the modules, and processing system interrupts.

When a network packet is received, an interrupt triggers a routine that copies the packet header to the feature extractor register bank. Next, the feature extractor module processes the received header and extracts the 54 features (Table 5) used by the classifiers. Finally, the classifiers analyze the feature values and label the incoming frame as normal or attack (Fig. 13). The output of each classifier is used as an input to a corresponding rejection logic block. Each rejection block uses the class value and the extracted feature values to determine whether the classifier output should be rejected.

The rejection limits are implemented as constants inside the blocks. When five or more of the normalized attribute values lie outside the allowed limits (Fig. 10, Average Rejection Point thresholds), the output of the rejection logic block is true, meaning that the class output should be rejected. The reject output of the combination classifier is true only when all the three individual reject outputs are true.

6. Experimental Results

We have measured the energy consumption and data throughput of each module, including the Packet Capture, Feature Extraction, and Detection Engine modules (the detection engine includes our proposed rejection method). To evaluate the packet capture module, the packets from our baseline scenario (Table 3) were sent using the Tcpreplay tool [66] running on a remote computer. To measure the power consumption of the software implementation, we used our power measurement platform (Fig. 12). To measure the consumption of the hardware implementation running on the FPGA, we used Altera's Power Monitor tool. This tool measures the FPGA consumption using onboard ADCs and sends the results continuously to a PC via JTAG.

The energy consumed per operation (capture, extraction or classification of each packet) was calculated with (2), where $P_{running}$ denotes the motherboard or FPGA power consumption while the algorithm is running, and P_{idle} denotes the motherboard or FPGA baseline power consumption. We discount the baseline consumption because our goal is to compare the two approaches. To calculate the throughput, we used (3).

The processing time in HW is calculated from the clock frequency (50 MHz for all circuits) and the number of clock cycles required to complete an operation.

$$Energy\ per\ operation\ (J) = \frac{(P_{running} - P_{idle}) * processing\ time}{number\ of\ packets} \quad (2)$$

$$Throughput\ (packets/s) = \frac{number\ of\ packets}{processing\ time} \quad (3)$$

Table 7 shows the measured power consumption, in SW and in HW, to capture and filter the packets from the network interface. Although they are functionally identi-

TABLE 7
POWER CONSUMPTION OF THE PACKET CAPTURE AND FILTER BLOCK

Implementation	Power (mW)
Packet capture and filter in SW	617.2
Packet capture and filter in HW	226.3

TABLE 8
ENERGY CONSUMPTION AND THROUGHPUT OF EACH MODULE

Algorithm	Energy (uJ/packet)		Throughput (packets/s)	
	SW	HW	SW	HW
Feature Extractor	1.873	1.078	363,875	534,815
Decision Tree	0.3809	0.0839	979,220	2,489,333
Naïve-Bayes	2.948	0.941	164,882	92,140
LDA	1.223	1.172	382,193	68,795
Combined classifier	4.271	1.540	111,641	66,452

cal, the HW implementation uses only 36.7% of the power used by the SW version (226.3 mW *vs* 617.2 mW). This difference in power consumption suggests that the HW version of this module can provide significant energy savings.

Table 8 shows the energy consumption and throughput of each module, including the feature extractor, the three single classifiers (DT, NB, and LDA), and the combined classifier. As for the energy spent to process a packet, all HW implementations require less energy than their SW counterparts do. The HW versions use between 22% (DT classifier) and 96% (LDA classifier) of the energy used by their SW equivalents. As for the throughput, only the DT classifier is faster in HW.

Table 9 shows the FPGA logic resources used by the HW implementation of each module. The area (in logic elements, or LEs) of the combination classifier (13,384 LEs) is approximately the same as the sum of the individual classifiers (13,495 LEs). The only classifier that requires dedicated memory blocks is LDA (36 kB). The combination classifier requires as many clock cycles as the slowest individual classifier (LDA, 606 cycles), and its maximum operating frequency (40.3 MHz) is slightly lower than that of the slowest classifier (LDA, 41.7 MHz).

7. RELATED WORKS

Over the past years, anomaly-based intrusion detection using classifiers has been extensively studied due to its ability to detect new attacks [45]. However, as noted in [3] and [25], there is a lack of real-world applicability in the works found in the literature. Although several studies present promising results and reasonably high detection rates [2], most do not deal with the practical aspects of network-based intrusion detection, such as detection throughput, traffic profile changes, and energy consumption [25].

Because real-world network traffic is highly variable and context-dependent [26], approaches modeling the traffic behavior through statistical methods [33] [34] will fail if they assume that the network traffic is static. Shiravi et al. [33] deal with changes in network behavior by creating abstract distribution models for applications and detailed descriptions of intrusions. In their approach, each client has a specific profile that was statistically modeled according to real network traces from several services. However, this approach leads to a database environment that is too specific and hard to maintain, because the environment must be reanalyzed for each model update.

Several authors have proposed the sanitization of real network traces in order to provide intrusion databases as realistic as possible [35] [36] [37] and to distribute them publicly. However, as noted in [46], sensitive data can still be extracted in spite of traffic sanitization processes. To overcome these problems, the approach used in our work obtains the data using well-known tools in a controlled environment. We mimic the profile changes by generating different scenarios with different traffic characteristics.

To the best of our knowledge, our work is the first to address the classifier reliability in the presence of traffic profile changes.

To minimize the energy consumption and maximize the system throughput, several authors use hardware-based implementations. Das et al. [19] developed an FPGA architecture composed of a feature extraction module and a detection module, using the Principal Component Analysis technique to detect port scan (probing) and syn flood (DoS) attacks. The throughputs achieved for extraction and detection were 21 and 23 Gbps, respectively, using a Xilinx Virtex-II XC2V1000. However, the authors used the outdated KDD'99 dataset to validate their implementation.

TABLE 9
LOGIC RESOURCES UTILIZATION AND TIMING PERFORMANCE OF THE HW CLASSIFIER IMPLEMENTATIONS

Classifier	LEs	Memory bits	Multipliers	#cycles	Fmax (MHz)
Decision Tree	1,247	0	0	45	112.0
Naïve-Bayes	3,640	36,864	14	486	44.8
LDA	8,608	0	14	606	41.7
Combination	13,384	36,864	28	606	40.3

A software and hardware comparison was performed in [17]; the authors showed that the hardware version of the decision tree algorithm consumed only 0.03% of the energy used by its software counterpart. In our previous work [18], energy savings were pursued at the feature extraction module by considering the energy required to extract the selected set of features. The subject of maintaining the classification reliability in hardware implementations was not considered in any work in the literature.

8. CONCLUSION

The growing number of embedded systems connected to the internet has increased the demand for security solutions that are energy efficient and provide classification reliability, detection accuracy, and throughput. Anomaly-based intrusion detection using machine-learning classifiers is an increasingly popular approach to detect network attacks.

In order to evaluate the applicability of the proposed solution, we conceived a testbed environment that mimics real-world network characteristics. Using our test scenarios, we presented a new rejection method that addresses network profile changes over time by defining

an assigned class $profile_{similarity}$ level according to the current feature values. By using our proposed approach, we were able to reject a small number of known and similar events (0.25% and 0.15% respectively) while still improving the classification accuracy by 13.98% when detecting new attacks.

To improve the classification accuracy and reduce the rejection rate, we evaluated a combination of various independent classifiers using a majority-voting scheme. The combination approach improved the classification accuracy by an average of 0.39% and 10.94% for similar and new profiles respectively, when compared to the single classifier approach. The combination approach provided improved reliability while still rejecting fewer instances.

Finally, we have evaluated a hardware implementation of the proposed solution that provides reliable intrusion detection and is more energy-efficient than the corresponding software version. Our hardware implementation (feature extractor plus combination classifier) consumed only 42.6% of the energy required by the corresponding software version (2.6 uJ vs. 6.1 uJ, respectively).

ACKNOWLEDGMENT

The authors would like to thank Intel Labs Univ. Research Office and the Brazilian National Council for Scientific and Technological Development (CNPq), grant 307346/2015-3, for the financial support.

REFERENCES

[1] OWASP, Internet of Things (IoT) Project. https://www.owasp.org/index.php/OWASP_Internet_of_Things_Top_Ten_Project, May 2018.

[2] C. F. Tsai, Y. F. Hsu, C. Y. Lin, and W. Y. Lin, "Intrusion detection by machine learning: A review," Expert Syst. Appl., vol. 36, no. 10, pp. 11994–12000, 2009.

[3] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," 2010 IEEE Symp. Secur. Priv., vol. 0, no. May, pp. 305–316.

[4] Jyothsna, V. V Rama Prasad, and K. Munivara Prasad, "A Review of Anomaly based Intrusion Detection Systems," Int. J. Comput. Appl., vol. 28, no. 7, pp. 26–35, 2011

[5] A. S. Britto, R. Sabourin, and L. E. S. Oliveira, "Dynamic selection of classifiers—A comprehensive review," Pattern Recognit., vol. 47, no. 11, pp. 3665–3680, 2014.

[6] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen, "Automatic recognition of handwritten numerical strings: A Recognition and Verification strategy," IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 11, pp. 1438–1454, 2002.

[7] H.-Y. Lam and D.-Y. Yeung, "A Learning approach to spam detection based on social networks," 4th Conf. Email anti-spam ({CEAS 2007}), pp. 81–89, 2007.

[8] L. S. Oliveira, R. Sabourin, F. Bortolozzi, and C. Y. Suen, "Impacts of verification on a numeral string recognition system," Pattern Recognit. Lett., vol. 24, no. 7, pp. 1023–1031, 2003.

[9] R. Perdisci, O. Gu, e W. Lee, "Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems", Proc. IEEE Int. Conf. Data Mining, ICDM, p. 488–498, 2006.

[10] G. Giacinto, R. Perdisci, M. Del Rio, e F. Roli, "Intrusion detection in computer networks by a modular ensemble of one-class classifiers", Inf. Fusion, vol. 9, no 1, p. 69–82, 2008.

[11] A. Zainal, M. A. Maarof, e S. M. Shamsuddin, "Ensemble Classifiers for Network Intrusion Detection System", Comput. Intell., vol. 4, no July, p. 217–225, 2009.

[12] D. Li, H. Zhou, e K. M. Lam, "High-resolution face verification using pore-scale facial features", IEEE Trans. Image Process., vol. 24, no 8, p. 2317–2327, 2015.

[13] M. Baena-Garcia, J. Del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early Drift Detection Method," 4th ECML PKDD Int. Work. Knowl. Discov. from Data Streams, pp. 77–86, 2006.

[14] M. R. Kmieciak and J. Stefanowski, "Semi-supervised approach to handle sudden concept drift," Control and Cybernetics vol. 40, no. 3, 2011.

[15] V. González-Castro, R. Alaiz-Rodríguez, and E. Alegre, "Class distribution estimation based on the Hellinger distance," Inf. Sci. (Ny)., vol. 218, pp. 146–164, 2013.

[16] M. Baena-Garcia, J. Del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, "Early Drift Detection Method," 4th ECML PKDD Int. Work. Knowl. Discov. from Data Streams, pp. 77–86, 2006.

[17] A. L. França, R. P. Jasinski, P. R. Cemin, V.A. Pedroni and A. O. Santin, "The Energy Cost of Network Security: a Hardware vs. Software Comparison," in Proc. IEEE ISCAS, 2015, pp 81-84.

[18] E. Viegas, A. Santin, A. França, R. Jasinski, V. Pedroni, and L. Oliveira, "Towards an Energy-Efficient Anomaly-Based Intrusion Detection Engine for Embedded Systems," IEEE Transactions on Computers, vol. 66, no. 1, pp. 1–14, 2017.

[19] Das, A.; Misra, S.; Joshi, S.; Zambreno, J.; Memik, G.; Choudhary, A., "An Efficient FPGA Implementation of Principle ComponentAnalysis based Network Intrusion Detection System," in DATE, pp.1160-1165, 2008.

[20] Embedded Intel solutions, "Intel's Hybrid CPU-FPGA," www.embeddedintel.com/commentary.php?article=2143. Accessed: May/2018.

[21] T. T. T. Nguyen e G. Armitage, "A survey of techniques for internet traffic classification using machine learning", Commun. Surv. Tutorials, IEEE, vol. 10, no 4, p. 56–76, 2008.

[22] G. Fumera and F. Roli, "Analysis of error-reject trade-off in linearly combined multiple classifiers", Pattern Recognit., vol. 37, no 6, p. 1245–1265, 2004.

[23] C.K. Chow, "On optimum error and reject tradeoff", IEEE Trans. Inform. Theory IT, 1970.

[24] G. Fumera, F. Roli, and G. Giacinto, "Reject option with multiple thresholds," Pattern Recognit., vol. 33, no. 12, pp. 2099–2101, 2000.

[25] C. Gates and C. Taylor, "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," Proc. 2006 Work. New Secur. Paradig., pp. 21–29, 2007.

[26] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," IEEE/ACM Trans. Netw., vol. 3, no. 3, pp. 226–244, 1995.

[27] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Ferná Ndez, and E. Vá Zquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," Comput. Secur., vol. 28, pp. 18–28, 2009.

[28] V. Jyothsna, V. V Rama Prasad, and K. Munivara Prasad, "A Review of Anomaly based Intrusion Detection Systems," Int. J. Comput. Appl., vol. 28, no. 7, pp. 26–35, 2011.

[29] Symantec Lab. ISTR20: Internet Security Threat Report, [online] available: www.symantec.com/content/en/us/enterprise/other_resources/2134

7933_GA_RPT-internet-security-threat-report-volume-20-2015.pdf. Accessed May/2018.

[30] C. Komviriyavut, T.; Sangkatsanee, P.; Wattanapongsakorn, N.; Charnsripinyo, "Network intrusion detection and classification with Decision Tree and rule based approaches," 9th Int. Symp. Commun. Inf. Technol. pp. 1046–1050, 2009.

[31] P. Gogoi, M. H. Bhuyan, D. K. Bhattacharyya, J. K. Kalita, "Packet and Flow Based Network Intrusion Dataset", Contemporary Computing, p. 322-334, 2012.

[32] E. K. Viegas, A. O. Santin, and L. S. Oliveira, "Toward a reliable anomaly-based intrusion detection in real-world environments," Comput. Networks, pp. 200-216, vol. 127, 2017.

[33] A. Shiravi, H. Shiravi, M. Tavallaee, and A. a. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," Comput. Secur., vol. 31, no. 3, pp. 357–374, 2012.

[34] 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., vol. 3, no. 4, pp. 262–294, 2000.

[35] CAIDA. The cooperative association for internet data analysis [online] available: http://www.caida.org/. Accessed May/2018.

[36] RTI International. PREDICT: Protected repository for the defense of infrastructure against cyber threats [online] available: http://www.predict.org. Accessed Nov./2015.

[37] Lawrence Berkeley Nationatiol Laboratory, The internet traffic archive [online] available: http://ita.ee.lbl.gov/index.html. Accessed Jun./2016.

[38] CISCO. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper [online] available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf. Accessed May/2018.

[39] "Honeyd." www.honeyd.org/.Accessed May/2018.

[40] S. Brugger and J. Chow, "An assessment of the DARPA IDS Evaluation Dataset using Snort," UCDAVIS Dep. Comput. Sci., pp. 1–19, 2007.

[41] E. Viegas, A. Santin, V. Abreu, and L. S. Oliveira, "Stream learning and anomaly-based intrusion detection in the adversarial settings," in Proceedings - IEEE Symposium on Computers and Communications, pp. 1-6, 2017.

[42] U. M. Fayyad and K. B. Irani, "Multi-Interval Discretization of Continuos-Valued Attributes for Classification Learning," Proceedings of the International Joint Conference on Uncertainty in AI. pp. 1022–1027, 1993.

[43] S. Mika, G. R¨atsch, J. Weston, B. Sch¨olkopf, and K.-R. Muller. ¨Fisher discriminant analysis with kernels. In Hu, Larsen, Wilson, and Douglas, editors, Proceedings of the IEEE Workshop on Neural Networks for Signal Processing IX, pages 41–48, 1999.

[44] "Libpcap" http://www.tcpdump.org/. Accessed Jun./2016.

[45] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," IEEE Trans. Syst. Man Cybern. Part C Appl. Rev., vol. 40, no. 5, pp. 516–524, 2010.

[46] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks," 2011 IEEE Symp. Secur. Priv., pp. 3–18, 2011.

[47] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," Comput. Secur., vol. 30, no. 6–7, pp. 353–375, 2011.

[48] "Weka." www.cs.waikato.ac.nz/ml/weka/. Accessed May/2018.

Eduardo Viegas received the BS degree in computer science from PUCPR in 2013 and is currently working toward the PhD degree 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 Computer Engineering at PUCPR. He is a member of the IEEE, ACM, and the Brazilian Computer Society.

André L. França received the BS degree in Electrical Engineering from Federal University of Parana (UFPR) in 2013 and the MSc degree in Electrical and Computer Engineering from Federal Technological University of Parana (UTFPR) in 2015.

Ricardo P. Jasinski was born in Curitiba, Brazil, in 1977. He received his B.S., M.S., and Ph.D. degrees in electrical engineering from the Federal Technological University of Parana in 2000, 2004, and 2014.

Volnei A. Pedroni received his BSc in Electrical Engineering from UFRGS, in 1975, and his MSc and PhD from Caltech, in 1990 and 1995, respectively. He has been since with the Electronics Engineering Dept. of Federal Technological University of Paraná State (UTFPR), in Brazil.

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 and head of the Graduate Program in Computer Science.