Ensembles of Decision Trees for Network Intrusion Detection Systems

Alexandre Balon-Perin
abalonpe@ulb.ac.be

Ecole Polytechnique
Université libre de Bruxelles
Brussels, Belgium

Björn Gambäck
gamback@idi.ntnu.no

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

***Abstract*—The paper discusses intrusion detection systems built using ensemble approaches, i.e., by combining several machine learning algorithms. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Network attacks can be divided into four classes: probe, remote to local, denial of service, and user to root. Each module of the ensemble designed in this work is itself an ensemble created by using bagging of decision trees and is specialized on the detection of one class of attacks. Experiments highlighted the efficiency of the approach and showed that increased accuracy can be obtained when each class of attacks is treated as a separate problem and handled by specialized algorithms. In all experiments, the ensemble was able to decrease the number of false positives and false negatives. However, some limitations of the used dataset (KDD99) were observed. In particular, the distribution of examples of remote to local attacks between the training set and test set made it difficult to evaluate the ensemble for this class of attacks. Furthermore, the algorithms need to be trained with specific feature subsets selected according to their relevance to the class of attacks being detected.**

***Keywords*-intrusion detection, ensemble approaches, bagging, decision trees, support vector machines.**

I. Introduction

Intrusion detection systems (IDSs) are monitoring devices that have been added to the wall of security in order to prevent malicious activity on a system. Here we will focus on network intrusion detection systems mainly because they can detect the widest range of attacks compared to other types of IDSs. In particular the paper discusses machine learning based mechanisms that can enable the network IDS to detect modified versions of previously seen attacks and completely new types of attacks [1].

Network IDSs analyse traffic to detect on-going and incoming attacks on a network. Additionally, they must provide concise but sound reports of attacks in order to facilitate the prevention of future intrusions and to inform the network administrators that the system has been compromised. Current commercial IDSs mainly use a database of rules (*signatures*), to try to detect attacks on a network or on a host computer. This detection method is presently the most accurate, but also the easiest to evade for experienced malicious users, because variants of known attacks (with slightly different signatures) are considered harmless by the IDS and can pass through without warning. New attacks and attacks exploiting zero-day vulnerabilities can also slip through the security net if their signatures are unknown to the IDS. A *zero-day vulnerability* is a software weakness unknown by the system developers, which potentially could allow an attacker to compromise the system. 'Zero-day' refers to the first day, day zero, that the vulnerability was observed.

In order for an intrusion detection system to be able to detect previously unseen attacks or variants of known attacks, there is a need for mechanisms allowing the IDS to learn by itself to identify new attack types. However, the problem is further complicated by the extreme requirements of robustness of the IDS. It must be able to detect all previously seen and unseen attacks without failure, it must never let an attack pass through unnoticed, and it must never deliver unwanted warnings when the traffic is in fact legitimate. Sommer and Paxson (2010) give a summary of the main challenges that machine learning has to overcome to be useful for intrusion detection [2].

Despite these constraints and challenges, several attempts have been made to build automatically adaptable intrusion detection systems using various machine learning algorithms. So far though, the machine learning classifiers trigger too many false alarms to be useful in practice. Part of the problem is the lack of labelled datasets to train the classifiers on. The only freely available labelled dataset is the KDD99 dataset [3] described below (Section III). To address these problems, new machine learning paradigms have been introduced in the field of intrusion detection, and in general the machine learning community has in recent years paid more attention to ensemble approaches, that is, to combinations of several machine learning algorithms.

Network attacks can be divided into four classes: probe, remote to local, denial of service, and user to root. Most previous machine learning-based solutions include a single algorithm in charge of detecting all classes of attacks. Instead, in this work, one module of an ensemble is specialised on the detection of attacks belonging to one particular class. The main idea is to exploit the strengths of each algorithm of the ensemble to obtain a robust classifier. Ensembles are

particularly efficient in cases like this, when a problem can be segmented into parts, so that each module of the ensemble is assigned to one particular subproblem. The modules in turn include one or more algorithms cooperating.

Furthermore, each class of attacks is characterized by very specific properties, observable through the values of certain features on instances in the dataset belonging to a specific class of attacks. However, even though feature selection is often applied in IDSs using machine learning techniques, often only one set of features is selected for *all* classes of attacks. In this work, one set of features is selected for *each* class of attacks according to their relevance to the corresponding class. The corresponding algorithm(s) is then fed with the appropriate set of features. The system can, in theory, reach a very high accuracy with a small cost, and the ensemble processing can potentially be parallelized using a multicore architecture. In the best scenario, each algorithm could run on a different core of the processor allowing the IDS to attain extremely high performance.

The experiments performed in this paper are in direct continuity of the work done by Mukkamala *et al.* [4]–[6], which identified the key features relevant to each of the four classes of attacks. The objectives of our experiments were multiple. In particular, to answer the following questions:

- Can ensemble approaches improve intrusion detection accuracy even when using the simplest algorithms without fine-tuning?
- Are the results of Mukkamala *et al.* (2005) [4] concerning the features selected by the three algorithms support vector machines (SVM), linear genetic programming (LGP), and multivariate adaptive regression splines (MARS), for each class of attacks, correct?
- Are the false positive (FP) and false negative (FN) rates close enough to zero for the IDS to be efficient?

The rest of the paper is laid out as follows: First, Section II introduces the machine learning methods utilized in the paper, in particular discussing ensembles and feature selection. Section III then discusses the data set used in the experiments, while Section IV gives an overview of the state-of-the-art and related work, in particular focusing on previous efforts in applying ensemble-based methods to intrusion detection. The core of the paper is Section V that details two rounds of experiments carried out, on feature selection for ensembles resp. on feeding an ensemble of machine learning algorithms with the most successful sets of features identified. Section VI then discusses the results of the experiments at length and points to ways in which the present work could be extended. Finally, Section VII sums up the previous discussion.

II. Ensemble-based Intrusion Detection

Machine learning algorithms operate in two main steps. In the first, the algorithm uses a training dataset to build a model of the data. In the second step, the model is applied to new examples. Usually, a test set is used to assess the performance of the algorithm. The model differs greatly depending on the type of algorithm used. In the case of regression, the algorithm must find the function that fits the data as well as possible. In the case of classification, the algorithm must find decision boundaries that separate the data as well as possible according to the number of desired classes. In both cases, a cost function is used to evaluate how good the model fits the data. The goal of the machine learning algorithm is to find the model that minimizes the cost function.

A. Supervised and Unsupervised Learning

In general, machine learning algorithms can be divided into two major classes depending on their learning technique: supervised and unsupervised. Supervised learning implies to obtain a training dataset in which every entry is labelled with class the example belongs to, while unsupervised learning algorithms do not need the dataset to be labelled. This is the most obvious disadvantage of supervised learning: obtaining data is cheap whereas obtaining labels for the data is very expensive in terms of both time and money because one or more experts must go through millions of examples and assign them a label. Apart from this main drawback, supervised learning also has some advantages. The first one is the ease of use and interpretation of the results. Indeed, the output of the classifier belongs to one of the classes defined by the labels of the dataset. The second advantage of supervised learning is its accuracy to classify similar examples. However, this accuracy drops significantly when the new examples are not so similar to the ones in the training set [7].

The most popular technique of unsupervised learning is clustering, where the algorithm exploits the similarity of the examples in order to form clusters or groups of instances. Examples belonging to the same cluster are assumed to have similar properties and belong to the same class. In contrast to supervised learning, disadvantages of unsupervised learning include manual choice of the number of cluster that the algorithm must form, lower accuracy of the prediction, and that the meaning of each cluster must be interpreted to understand the output. However, unsupervised learning is more robust to large variations. This is a very important advantage when applied to the problem of intrusion detection, since it means that unsupervised learning is able to generalize to new types of attacks much better than supervised learning. In particular, this property could be quite beneficial when trying to detect zero-day vulnerabilities.

B. Ensembles

The ensemble method is a way to build different types of approaches to solving the same problem: the outputs of several algorithms used as predictors for a particular problem are combined to improve the accuracy of the overall system. Ensemble approaches were introduced for the first time in the late 80s. In 1990, Hansen and Salamon showed that

the combination of several artificial neural networks can drastically improve the accuracy of the predictions [8]. The same year, Schapire showed theoretically that if weak learners (i.e., classifiers able to correctly classify only a small fraction of the examples in a dataset) are combined, it is possible to obtain an arbitrary high accuracy [9].

The difficulty of ensemble approaches lays in the choice of the algorithms constituting the ensemble and the decision function that combines the results of the different algorithms. Often, the more algorithms the better, but it is important to take into account the computational expense added by each new algorithm. The decision function is often a majority vote that is both simple and efficient, but alternatives should be analysed to obtain an optimal combination. Another advantage of ensemble approaches is their modular structure, unlike hybrid constructions that are engineered with algorithms having non-interchangeable positions. Consequently, the ensemble designer can easily replace one or more algorithms with a more accurate one.

Bagging and boosting are the two main techniques used to combine the algorithms in an ensemble. In an ensemble using the *boosting* technique, the algorithms are used sequentially. The first algorithm analyses all the examples in the dataset and assigns weights to each of them. The examples with a higher value for the weight are the ones that were classified wrongly by the algorithm. Then, the next algorithm receives as input the dataset as well as the weights for all examples in the dataset. The weights allow the algorithm to focus on the examples that were the most difficult to classify. These weights are updated according to the results of the second algorithm and the process moves to the third algorithm. This sequence continues until the last algorithm of the ensemble has processed the data. The advantage of this technique is that the most difficult examples can be classified correctly without adding too much computational overload. The use of weights, which are continuously updated, reduces the processing time as the data goes down the chain of algorithms.

In an ensemble using the *bagging* technique, all algorithms of the ensemble are used in parallel. In this case, each algorithm builds a different model of the data and the outputs of all predictors are combined to obtain the final output of the ensemble. In order to build different models, either each algorithm of the ensemble, or the data fed to each algorithm, or both, can be different. Since all algorithms perform in parallel, each of them can be executed on a different processor to speed up the computation. This is an important advantage over the boosting technique because nowadays multicore processors are very common even on personal computers. With this kind of architecture, the ensemble does not significantly increase the processing time compared to a single algorithm because the only additional time needed is used for the decision function that combines the outputs of all algorithms.

C. Feature Selection

Feature selection is a very efficient way to reduce the dimensionality of a problem. Redundant and irrelevant variables are removed from the data before being fed to the machine learning algorithm used as a classifier. Feature selection is a preprocessing step that commonly is independent of the choice of the learning algorithm. It can be used in order to improve the computational speed with minimum reduction of accuracy. Other advantages include noise reduction and robustness against over-fitting since it introduces bias but drastically reduces the variance. Generally, automatic selection of features works much better than manual selection because the algorithm is able to find correlations between the features that are not always obvious even for a human expert. Feature selection is an important preprocessing step of a machine learning algorithm that should not be overlooked. In particular, it should always be applied when the problem has a high dimensionality, as is the case for intrusion detection, since there is no point in feeding an algorithm with features that are irrelevant or add an insignificant amount of new information.

The main feature selection algorithms are minimum redundancy maximum relevance (mRMR) and principal component analysis (PCA). The former selects the subset of variables most relevant to the problem. The variables are ranked according to the information that they contain. This quantity of information is calculated by using the concept of entropy from information theory. The latter, PCA transforms the set of variables into a new smaller set of features. In both cases, the goal is to extract as much information as possible from as few features as possible. While PCA has been extensively used for the problem of intrusion detection, particularly on the KDD99 dataset, surprisingly, mRMR seems not to have been used much or at all [10].

III. The KDD99 Dataset

As observed in the introduction, part of the problem of automatically creating good intrusion detection systems is the lack of labelled datasets to train on. The only one freely available is the KDD Cup 99 dataset, which was used for the first time in the 3rd International Knowledge Discovery and Data Mining Tools Competition in 1999. It is an adaptation of the DARPA98 dataset [11] created in 1998 by the (then) Defense Advanced Research Projects Agency (DARPA) Intrusion Detection Evaluation Group (now the Cyber Systems and Technology Group of MIT Lincoln Laboratory). The DARPA98 set includes seven weeks of data (captured in the form of a tcpdump) from traffic passing through a network engineered for the purpose, i.e., the traffic was generated in a simulated and controlled environment.

A few alternative datasets exist, but are limited by either not being generally accessible to the research community or by not being annotated. The UNB ISCX Intrusion Detection Evaluation DataSet [12] from the Information Security Centre

of eXcellence at University of New Brunswick contains more realistic/real network traffic data than the KDD99 dataset (i.e., mainly normal traffic, with just some intrusion attempts). However, it seems unfortunately currently not to be available to other researchers. Publically available datasets from, for example, the Internet Traffic Archive (http://ita.ee.lbl.gov) and University of New Mexico (http://www.cs.unm.edu/~immsec) contain a lot of data, but unannotated. Tavallaee *et al.* (2010) give an overview of most currently available datasets [13].

A. Types of Attacks

All the examples in the KDD99 dataset are separated into five classes: `Normal`, `Probe`, `R2L` (remote to local), `DoS` (denial of service), and `U2R` (user to root). The class `Normal` of course denotes normal (legitimate) network traffic. The other four classes denote different types of attacks (intrusion attempts) and are further described in turn below.

`Probe` attacks are scouting missions used to gather information about the targeted network or a specific machine on a network: attackers scan a network to find vulnerabilities and to create a map of the network, often as the first step of one of other types of attacks. Hence it is crucial to detect this type of attacks. However, it is difficult to differentiate attacks from regular actions, since probing or scanning typically abuse perfectly legitimate features used by network administrators to check on machines in a network. The most common program to scan a network is 'nmap', which can be used to look for active machines and active ports on a machine, or to discover the type and version of the server and the operating system. Other probes such as 'saint' and 'satan' are specialised in discovering vulnerabilities in the targeted system.

In `R2L` attacks, external attackers start a session on a computer outside of the targeted network and then manage to exploit some vulnerability in a system in order to get local user access on a computer in the network. In order to do this, the attackers must have the ability to send network packets to the victim host. Many remote to local attacks (e.g., 'warezmaster', 'warezclient', 'imap', 'named', and 'sendmail') exploit bugs or weaknesses in different Internet protocols such as FTP, DNS, and SMTP. Other `R2L` attacks exploit system misconfigurations (e.g., 'dictionary', 'ftp_write', 'guest', and 'xsnoop').

`DoS` attacks aim either to overload a system so that it cannot process all requests, or to directly deny legitimate users access to a system or network resource, such as network bandwidth, computer memory or computing power. An attacker can abuse a legitimate feature of a network protocol by, for example, sending replies to protocol queriers faster than the destination of the query in order to falsify the information contained in the network tables of the victim. Some of these attacks are 'mailbomb', 'neptune', 'smurf', and 'ARP poisoning'. Others such as 'teardrop' and 'ping of death' ('pod') exploit implementation bugs of the TCP/IP protocol, while attacks like 'apache2', 'back', and 'syslogd' target a specific program running on the victim host.

In the type of `DoS` attacks focusing on resource exhaustion, the attacker typically sends a huge amount of queries in a short amount of time to the targeted victim. If the victim is a server, resource exhaustion occurs when the server receives more queries than it can process: in a 'udpstorm' (also called 'UDP Port DoS' attack or 'UDP packet storm'), an attacker forges a packet with a spoofed source address of a host running an "echo" or "chargen" process and sends it to another hosts running a similar process. The receiving host replies with an echo packet to the spoofed source, which replies with another echo packet, etc., creating a loop leading to resource exhaustion or performance degradation [14].

A variant of `DoS` used extensively by hackers is distributed denial of service (DDoS) [15], [16]. A DDoS is performed in two main steps. First, an attacker gains control over a (often huge) number of computers, called slaves or zombies, by exploiting unpatched vulnerabilities found in the target systems. Then the attacker orders all slaves to query a designated machine (usually a server) at the same time.

In `U2R` attacks, access to a normal user account (with restricted rights) is used as a starting point to gain root user permissions and take over a system, e.g., by exploiting some vulnerability in the system. There are several different types of user to root attacks, with the most common being 'buffer overflow' [17] that aims to corrupt a program with high privileges (i.e., root) in order to take control of the host computer running the vulnerable program. The attacker uses a buffer with non-existent or poorly performed boundary checking to launch a root shell and then corrupts the stack pointer to point to the attacker's own malicious code. Other `U2R` attacks such as 'loadmodule' or 'perl' take advantage of the way some programs sanitize their environment. Others still (e.g., 'ps') exploit poor management of temporary files.

Table I
DISTRIBUTION OF INSTANCES IN THE KDD99 DATASETS

Class	Training set	Test set
`Normal`	972,781	60,593
`Probe`	41,102	4,166
`R2L`	1,126	16,347
`DoS`	3,883,370	229,853
`U2R`	52	70
Total	4,898,431	311,029

B. Training and Test Sets

The KDD99 dataset is divided into a training set and a test set. Table I shows the distribution of instances of the KDD99 training and test sets over the different classes of attacks. The various types of attacks belonging to each of these classes are further detailed in Table II.

Table II
TYPES OF ATTACKS IN THE KDD99 DATASETS

Class	Attack	Training set	Test set
Probe	satan	15,892	1,633
	ipsweep	12,481	306
	portsweep	10,413	354
	nmap	2,316	84
	mscan	0	1,053
	saint	0	736
	Total	41,102	4,166
R2L	warezclient	1020	0
	guess_passwd	53	4,367
	warezmaster	20	1,602
	imap	12	1
	ftp_write	8	3
	multihop	7	18
	phf	4	2
	spy	2	0
	snmpgetattack	0	7,741
	snmpguess	0	2,406
	httptunnel	0	158
	named	0	17
	sendmail	0	17
	xlock	0	9
	xsnoop	0	4
	worm	0	2
	Total	1,126	16,347
DoS	smurf	2,807,886	164,091
	neptune	1,072,017	58,001
	back	2,203	1,098
	teardrop	979	12
	pod	264	87
	land	21	9
	mailbomb	0	5,000
	apache2	0	794
	processtable	0	759
	udpstorm	0	2
	Total	3,883,370	229,853
U2R	buffer_overflow	30	22
	rootkit	10	13
	loadmodule	9	2
	perl	3	2
	ps	0	16
	xterm	0	13
	sqlattack	0	2
	Total	52	70
Normal		972,781	60,593
Total		4,898,431	311,029

Each entry in the sets is represented by a label and 41 features such as `duration`, `src_bytes`, and `dst_bytes`. Of the features, 38 are numerical and thus only three non-numerical: `protocol_type`, `service`, and `flag`. For the non-numerical features, there are three protocol types (`TCP`, `UDP`, and `ICMP`), 70 different services, and 11 possible flags. The non-numerical variables are normally transformed into numerical ones to ensure that all the machine learning algorithms are able to process their values.

The KDD99 training set contains $4,898,431$ entries and is highly unbalanced. Whereas the `DoS` class contains $3,883,370$ instances, the classes `U2R` and `R2L` are represented by only 52 and $1,126$ instances, respectively. With such a small number of examples to train on, it can be expected that it will be difficult for the classifiers to predict the correct classes of unseen examples.

The test set is composed of $311,029$ entries with a distribution of the examples over the different classes similar to that in the training set. However, the number of examples belonging to the class `R2L` is more than ten times higher than in the training set, so that in order to perform well on the test set, the predictor must acquire a very high power of generalisation with $1,126$ training examples. Most importantly, the number of unseen attacks added in the test set is huge: for the classes `U2R`, `R2L` and `Probe`, it is respectively $44.29\,\%$, $63.34\,\%$ and $42.94\,\%$. Furthermore, the attacks 'spy' and 'warezclient' belonging to the class `R2L` are not represented in the test set. In particular, 'warezclient' attacks count for more than $90\,\%$ of the `R2L` training set.

Notably, two entries in the test set erroneously have a `service` value of `ICMP`, as also previously reported [18]. Those faulty entries were removed from the test set before carrying out the experiments reported in Section V.

The major criticisms of the KDD99 dataset include the unbalanced distribution of the data, that the redundant records can introduce a bias in the learning phase because of their frequency, that the dataset includes old attacks which have been mostly mitigated, and that the data were captured from a controlled environment somewhat different from what is observed in the wild. The first two issues can be addressed by sampling appropriate sets of examples for each class of attacks. However, the distribution of `R2L` attacks in the training set and the test set is a problem that is difficult to overcome.

Nevertheless, the KDD99 dataset is far from useless. Firstly, if an IDS using machine learning does not perform well on old attack provided that the data are well sampled, why would it on newer ones? Furthermore, most of the research in the field of machine learning applied to intrusion detection uses the KDD99 dataset, making it a vector of comparison between different approaches. The controlled nature of the environment in which the data were captured is probably the most problematic. For example, the high number of attacks in comparison to normal traffic does not reflect the reality of a network in which almost all traffic is normal. Again, appropriate sampling is required. In addition, the IDS should at least be accurate on data produced by a simulated environment before being tested on a real network where the traffic pattern is probably less predictable.

IV. Related Work

Intrusion detection systems have been around since the 80s. In 1980, Anderson introduced the concept of host-based intrusion detection [19]. Seven years later, Denning laid the foundations of intrusion detection system development [20]. Network-based intrusion detection systems were introduced in 1990 [21]. In the late 90s, researchers in artificial intelligence started to investigate applying machine learning algorithms to improve intrusion detection.

An intrusion detection system should be able to autonomously recognize malicious actions in order to defend itself against variants of previously seen attacks and against attacks exploiting zero-day vulnerabilities. Misuse-based IDSs can only detect attacks whose signatures are available in their signature database. Signatures of attacks are very specific, and a slight variation of the attack can make it unnoticeable for the IDS. That is why learning mechanisms must be implemented to detect and prevent these attacks without having to wait for an update of the signature database or a patch for the vulnerable system. Still, machine learning algorithms are designed to recognize examples similar to those available in the training set used to build the model of the data. Consequently, an IDS using machine learning would have a hard time detecting attacks which patterns are totally different from the data previously seen. In other words, even though machine learning is a suitable candidate to detect variants of known attacks, detecting completely new types of attacks might be out of reach for these kinds of algorithms.

For a summary of most research involving machine learning applied to IDSs until 2007, see Wu & Banzhaf (2010) who cover a range of techniques, including fuzzy sets, soft computing, and bio-inspired methods such as artificial neural networks, evolutionary computing, artificial immune systems, and swarm intelligence; comparing the performance of the algorithms on the KDD99 test set and showing that all algorithms perform poorly on the `U2R` and `R2L` classes [22]. The best results reported are by genetic programming with transformation functions for `R2L` and `Probe` and by linear genetic programming (LGP) for `DoS` and `U2R` (with $80.22\,\%$, $97.29\,\%$, $99.70\,\%$ and $76.30\,\%$ accuracy, respectively). However, since ensemble-based methods are fairly new in being applied to intrusion detection, the description of them in the review is somewhat limited. The first works on the topic date from 2003 and many papers were written in 2004 and 2005; recently (from 2010 onwards), there has been a renewed interest of ensembles in this field.

Abrahams *et al.* have performed several types of ensemble-based experiments, all on a subset of the DARPA98 dataset composed of $11,982$ randomly selected instances from the original dataset with a number of data for each class proportional to the size of the class, except for the smallest class which was included entirely. This data was then divided into a training set of $5,092$ and a test set of $6,890$ instances.

First, in [23], an ensemble composed of different types of artificial neural networks (ANN), support vector machines (SVM) with radial basis function kernel, and multivariate adaptive regression splines (MARS) combined using bagging techniques was compared to the results obtained by each algorithm executed separately. SVM used alone outperformed the other single algorithms, but was totally outperformed by the ensemble. This ensemble surprisingly obtained a $100\,\%$ accuracy on the test set for the `R2L` class. However, the researchers warn that some of these results might not be statistically significant because of the unbalanced dataset.

Second, in [24], [25], the combination of classification and regression trees (CART) and Bayesian networks (BN) in an ensemble using bagging techniques was explored, as well as the performance of the two algorithms when executed alone. Feature selection was applied to speed up the processing: the performance on the set of 41 features was compared to a set of 12 selected by BN, 17 selected by CART and 19 features selected by another study. BN performed worse with a smaller set of features except on the `Normal` class. However, when using the set of 19 features, BN and CART complemented each other to increase the IDS accuracy for all classes. The final ensemble was composed of three CART to detect `Normal`, `Probe` and `U2R` examples, respectively; one ensemble of one CART and one BN to detect `R2L` examples; and one ensemble of one CART and one BN to detect `DoS` examples — with each classifier trained on its resp. reduced set of features; an approach quite similar to the one used in the present paper.

This was then extended by adding a hybrid model composed of SVM and decision trees (DT) to the ensemble [26], [27]. In the new model, the data was first sent to the DT that generated a tree to fit the features and values of each example in the dataset. The tree was then sent to the SVM to produce the final output. A single DT was in charge of detecting `U2R` attacks, a single SVM in charge of detecting `DoS` attacks, the hybrid model in charge of `Normal` instances, and the same ensemble as above in charge of `Probe` and `R2L` attacks. However, the hybrid model did not seem to help much.

Third, in [28], [29], fuzzy rule-based classifiers, linear genetic programming (LGP), DT, SVM, and an ensemble were evaluated using feature selection to reduce the number of variables of the dataset to 12 on a subset of the DARPA 1998 dataset, selected in the same way as in the work mentioned previously. The fuzzy rule-based classifier outperformed the other methods when trained on all 41 features, with the second set of rules scoring 100 % accuracy for all classes of attacks; while LGP seemed more appropriate when using a smaller feature set, except for the U2R and Normal classes. The ensemble was composed of one DT in charge of the Normal instances, one LGP each for Probe, R2L and DoS, and one fuzzy set of rules for U2R. The results obtained with the ensemble were very encouraging with accuracy $> 99\,\%$ for all classes (on the subset data).

Finally, the results of several machine learning algorithms were compared in [30]. In particular, the performance of linear genetic programming (LGP), adaptive neural fuzzy inference system (ANFIS), and random forest (RF) were analysed, and an ensemble was created by combining the LGP, ANFIS, and RF algorithms. The ensemble outperformed the single algorithms, but its exact configuration is not described in the paper.

Folino *et al.* [31], [32], instead used the entire KDD99 dataset and examined the performance of a system composed of several genetic programming ensembles distributed on the network based on the island model. Each ensemble was trained on a different dataset for a number of rounds. Once the ensemble had been trained for one round, it was exchanged with the other islands through the distributed environment. The advantage of a distributed system, as pointed out by Folino *et al.*, is the increase in privacy and security in comparison to a central IDS that has to collect audit data from different nodes on the system. The system showed average performance for the Normal, Probe and DoS classes, but very low for the U2R and R2L classes. However, very few papers study distributed environment for intrusion detection even though this might be a very good idea.

Bahri *et al.* [33] introduced Greedy-Boost, a noise resistant adaptation of the AdaBoost boosting technique [34]. The Greedy-Boost classifier contrasts with AdaBoost by being a linear combination of models and by updating the distribution of weights according to the initial distribution instead of the previous one. Greedy-Boost's performance in terms of precision and recall on the KDD99 dataset was extraordinary good. In particular, the precision of the most difficult class (R2L) was much higher than what is usually observed. However, it is not clear from the paper if the model was evaluated on the test set, the training set, or a modified version of one of the sets.

Peng Zhang *et al.* [35] evaluated the robustness of an ensemble when confronted with "noisy" data sets, that is, data sets containing incorrectly labelled instances. In order to tolerate label imprecision and errors, they used an aggregate ensemble of SVM, DT, and logistic regression. An aggregate ensemble builds several classifiers over a range of data sets using different learning algorithms. The aggregate approach was shown to outperform both a horizontal ensemble framework in which classifiers were built over different data sets with only one learning algorithm for each set, and a vertical ensemble framework in which several classifiers were built over the data sets using different learning algorithms that were then combined into an ensemble.

The key conclusion from all these works is that ensemble approaches generally outperform approaches in which only one algorithm is used. An ensemble is a very efficient way to compensate for the low accuracy of a set of weak learners. Moreover, feature selection should provide specific subsets to train algorithms specialised in the detection of one particular class of attacks.

Mukkamala *et al.* [4]–[6] identified the five most important features for each class of attacks, as shown in Table III. The features were selected using support vector machines, linear genetic programming, and multivariate adaptive regression splines, with in total 16 distinct features selected for SVM,

Table III
MOST RELEVANT FEATURES FOR EACH ATTACK CLASS IN THE KDD99 DATASET ACCORDING TO MUKKAMALA *et al.* (2005) [4]

SVM features	LGP features	MARS features
	Probe	
src_bytes	srv_diff_host_rate	src_bytes
dst_host_srv_count	rerror_rate	dst_host_srv_count
count	dst_host_diff_srv_rate	dst_host_diff_srv_rate
protocol_type	logged_in	dst_host_same_srv_rate
srv_count	service	srv_count
	U2R	
src_bytes	root_shell	dst_host_srv_count
duration	dst_host_srv_serror_rate	duration
protocol_type	num_file_creations	count
logged_in	serror_rate	srv_count
flag	dst_host_same_src_port_rate	dst_host_count
	R2L	
srv_count	is_guest_login	srv_count
service	num_file_access	service
duration	dst_bytes	dst_host_srv_count
count	num_failed_logins	count
dst_host_count	logged_in	logged_in
	DoS	
count	count	count
srv_count	num_compromised	srv_count
dst_host_srv_serror_rate	wrong_fragments	dst_host_srv_diff_host_rate
serror_rate	land	src_bytes
dst_host_same_src_port_rate	logged_in	dst_bytes
	Normal	
dst_bytes	dst_bytes	dst_bytes
dst_host_count	src_bytes	src_bytes
logged_in	dst_host_rerror_rate	logged_in
dst_host_same_srv_rate	num_compromised	service
flag	hot	hot

21 for LGP and 13 for MARS. Features that are selected by at least two different algorithms for the same class of attacks are highlighted because they should most definitely be in the subset of features used to detect that specific class of attack.

Surprisingly, neither `protocol_type` nor `service` was selected by the three algorithms for the `DoS` class in the experiments by Mukkamala *et al.* (2005) [4]. In contrast, Kayacik *et al.* (2007) [36] concluded that those two features were the most significant ones for the denial of service class of attacks, even though the experiments by Kayacik *et al.* were conducted on hierarchical self-organizing maps (SOM).

V. Experiments

The problem of intrusion detection can be divided into five distinct subproblems, one for each class of instances (`Normal` and the four types of attacks: `Probe`, `U2R`, `R2L`, and `DoS`). Here each problem will be handled by one or more algorithms of an ensemble, allowing each subproblem to be treated separately in the experiments and to join the solutions to the subproblems into a general solution for the problem of intrusion detection.

A dedicated training dataset for each attack subproblem was built by sampling a number of instances from that class of attacks and the same number from the class `Normal` in order to have a balanced dataset with $50\,\%$ anomalous and $50\,\%$ normal examples (no algorithm was explicitly designed to detect normal traffic). A balanced dataset is necessary to avoid the problem of skewed classes where the accuracy of the predictor can be made artificially high by increasing the number of instances from one of the classes.

For the classes of attacks with few examples, `R2L` and `U2R`, the entire set was selected (i.e., 52 instances of `U2R` and $1,126$ of `R2L`). For the `Probe` class, $10,000$ instances were selected randomly. This number was chosen to have a significant sample with as many different examples as possible without affecting the training time too much. The `DoS` training set contains $3,883,370$ instances, with 'neptune' and 'smurf' attacks counting for the majority (with $1,072,017$ and $2,807,886$ instances, respectively). The other types of attacks have much smaller number of examples, e.g., the type of `DoS` called 'land' is represented only 21 times. For this reason, samples of $5,000$ examples each were selected randomly from the 'neptune' and 'smurf' sets. All examples of the other types of `DoS` attacks were included for a total of $13,467$ `DoS` instances. For all four classes, the same number of `Normal` instances was selected from the normal dataset leading to a total training set size of 104 examples for `U2R`, $2,252$ for `R2L`, $20,000$ for `Probe`, and $26,934$ for `DoS`.

In order to investigate the applicability of ensemble-based approaches to intrusion detection, two sets of experiments were carried out. The first step of the experiments was to assess the sets of features selected in [4], that is, the key features relevant to each of the four classes of attacks. Then in a second round of experiments, those sets were fed to an ensemble of machine learning algorithms. All models were evaluated by 10-fold cross-validation.

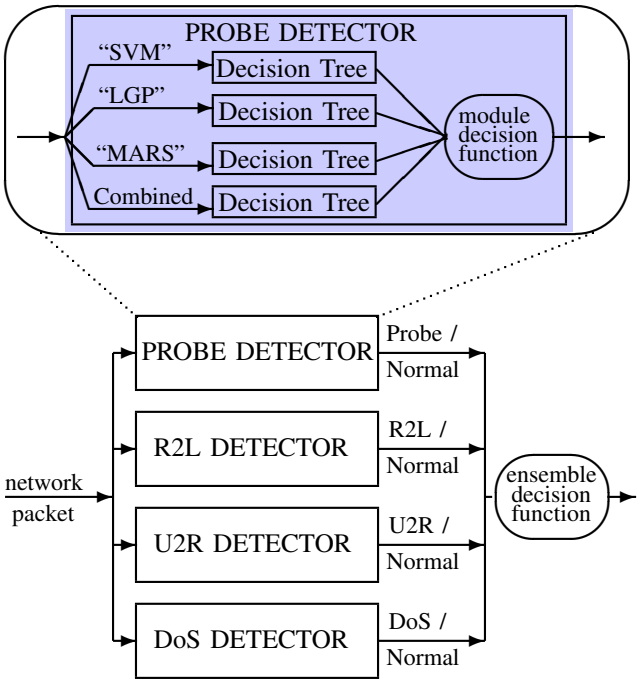

Figure 1. Overview of the ensemble model

A. Experimental Setup

Figure 1 gives a schematic overview of the ensemble model used in this work. First, the network packet being analysed is sent to four different detector modules, one each for `Probe`, `R2L`, `U2R`, and `DoS`. Each module executes a preprocessing step to extract a number of features from the packet; the set of features varies depending on the module (as further described in Section V-B). The extracted features are then dispatched to different decision trees that have been previously trained with the same features on the training set, as shown at the top of the figure for the `Probe` detector. Each decision tree is a binary classifier which outputs 0 if the packet is considered normal traffic and 1 if the packet is classified as anomalous. A vector of dimension n containing the output of n classifiers is then fed to the module decision function. In the figure $n = 4$ (one each for the features selected by SVM, LGP and MARS plus the set of those features combined), but it could be any number of algorithms.

Finally, a vector of dimension 4 containing the output of each detector module is fed to the ensemble decision function that combines the results and outputs a value describing if the packet is considered normal or anomalous, and if anomalous from which class of attacks. The easiest situations are when all modules, or all modules except one, output `Normal`. In the former case, the system classifies the packet as normal. In the latter, the system classifies the packet

as anomalous and is able to unambiguously identify the class of attack concerned. If more than one module classify the packet as anomalous, it will be more difficult for the network administrator to understand which class of attack the anomalous packet belongs to.

The resulting model is an ensemble of ensembles with feature selection applied independently for each module. However, in this work, we will not be concerned with the decision functions for each module. Instead, we will evaluate the intersection of the sets of false positives and false negatives produced by the four algorithms in each module. This will give us the optimal performance that each module could achieve. The most important advantages of this model are the possibility to execute the algorithms in parallel and the modularity allowing the exchange of any algorithm of the ensemble without any modification of the rest. The complete ensemble model is as shown in Figure 2 (on the next page).

B. Feature Selection Assessment with Decision Trees

In the first experiment, several classifiers were trained with different number of features. The goal of the experiment was not to find the best algorithm possible and fine-tune it, but rather to conclude on how well an algorithm performs with a smaller set of features. In this case, it is only natural to use exactly the same setting for the algorithms and to compare the performance based only on the sets of features. Five decision trees were trained with different sets of features. Only the training set was used for this experiment. The results obtained represent the performance of the algorithms on the cross-validation set which is extracted from the training set. The model assessment experiment (described below, in Section V-C) evaluated performance on the test set.

The first classifier was trained with all 41 features in the dataset. The next three were trained with five features selected in [4] for each class of attacks by the three algorithms support vector machines (SVM), linear genetic programming (LGP) and multivariate adaptive regression splines (MARS). These features are listed in Table III. The last classifier was trained on a "combined" set of features: the union of the feature sets selected by the three algorithms from which redundant features have been removed. The number of features in the "combined" set is 11 for `Probe`, 14 for `U2R`, 11 for `R2L` and 12 for `DoS`, as can be seen in Figure 2. These additional sets help bringing down the number of false positives and false negatives, as we will see in the results of the experiments. Note that there is no extra cost from the extraction of these features from the original network packets since they have to be extracted for the other algorithms anyway.

The results obtained in terms of accuracy are shown in Table IV and can be compared to those obtained with 41 features by Peddabachigari *et al.* (2007) [27] using the same decision tree. For the class `Probe`, the accuracy of the classifier trained with all 41 features is exactly the same as reported in [27]: 99.86 %. The classifiers trained with sets of five features are not far behind the one trained with all 41 features. The reduced feature sets seem to be a good choice when the algorithms are trained using decision trees. However, the classifier fed with the five features selected by LGP performs slightly worse than the others and could be replaced by a more accurate algorithm.

Table IV
ACCURACY OF THE FEATURE SELECTION ASSESSMENT

Decision Tree Classifier	`Probe`	`U2R`	`R2L`	`DoS`
41 features	99.86	93.00	99.02	99.95
5 SVM features	99.82	96.00	98.58	93.35
5 LGP features	99.32	90.00	97.38	98.69
5 MARS features	99.75	97.00	98.04	99.86
11–14 combined features	99.90	96.00	98.93	99.95
Peddabachigari *et al.* (2007) [27]	99.86	68.00	84.19	96.83

The results for `U2R` are worse than for `Probe`, but this was expected: each false positive and false negative has a larger impact on the general accuracy due to the small number of examples. The results (93–97 %) are much better than the 68 % accuracy obtained by Peddabachigari *et al.* on `U2R`. However, the classifier trained on features selected by LGP again performed poorly. Interestingly, the algorithms trained on the features selected by SVM and MARS outperformed the one trained on all features. This is probably since 41 features are too many to generalize from given the small number of examples; recall that the training set for `U2R` only held 52 instances (cmf. Table I).

The results for `R2L` are similar to those obtained for `Probe`, even though the number of instances in the dataset is much smaller. The results (97–99 %) are also much better than Peddabachigari *et al.* who obtained 84 % accuracy on this class. This experiment clarifies that classifying `Probe` attacks and `R2L` attacks are two very distinct problems, even if they are both intrusions, which is why they should be treated separately. Again, the selected features seem to be a good choice even if a small drop of accuracy can be observed compared to `Probe`. The classifier trained on the features selected by MARS has a high rate of false positives and the one trained on features selected by LGP has the lowest accuracy, but also a lower false positive rate, which implies a higher false negative rate.

`DoS` also shows better results than Peddabachigari *et al.* who obtained 96.83 % accuracy. The classifier trained on features selected by SVM obtained the worst score, whereas features selected by MARS gave the best accuracy (99.86 %) after the set of all features and the combined feature set that both reached 99.95 %. This is important, since there is a set of five features that can perform almost as well as the full feature set even on larger number of training examples.

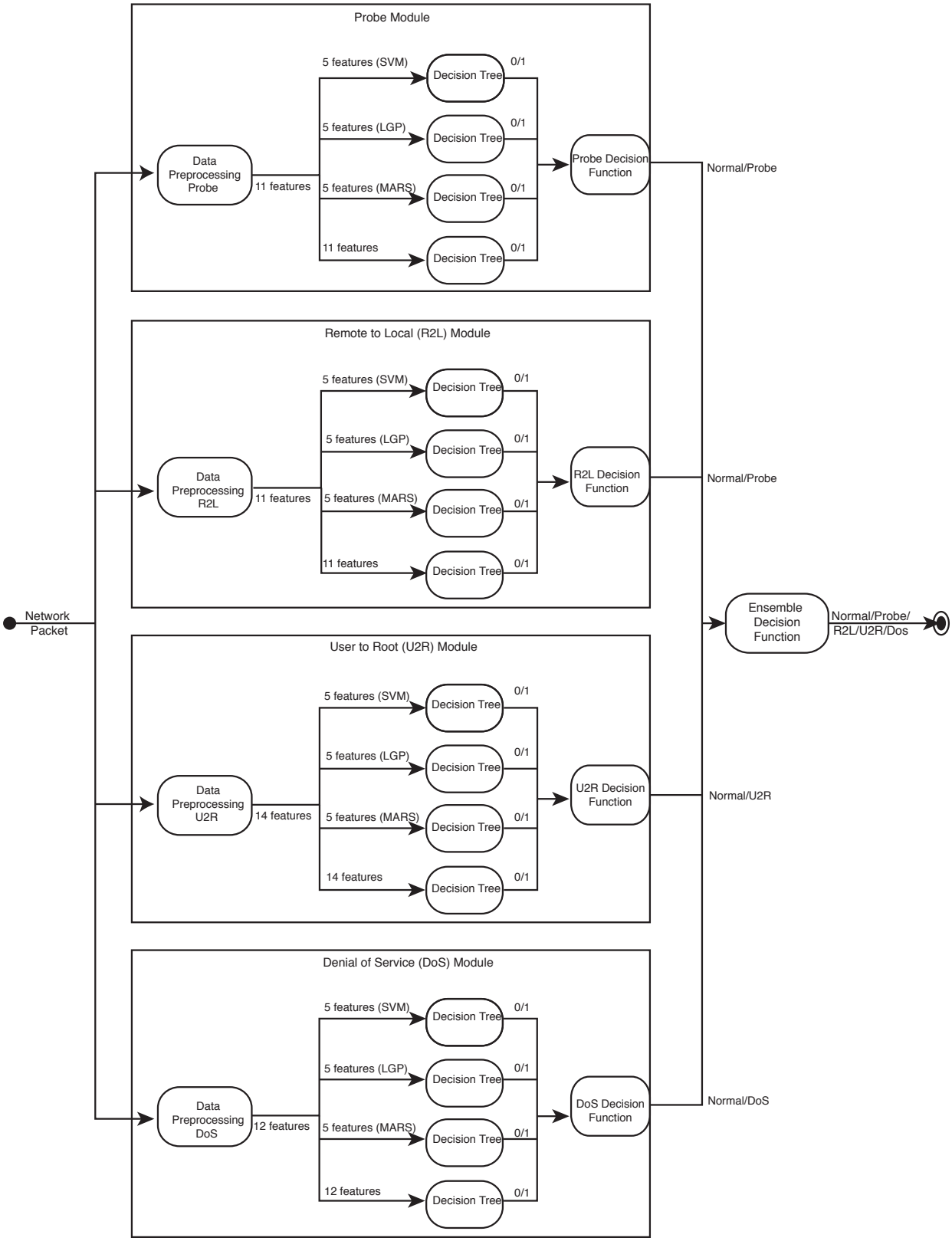

Figure 2. The complete ensemble. Each algorithm is a binary classifier outputting 0 if the packet is considered normal traffic and 1 if anomalous.

Table V
FEATURE SELECTION ASSESSMENT: FALSE POSITIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	12.0	4.0	17.0	6.0
ENSEMBLE$_{max}$	0.7	0.3	6.6	0.0

Table VI
FEATURE SELECTION ASSESSMENT: FALSE NEGATIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	17.0	3.0	10.0	8.0
ENSEMBLE$_{max}$	3.0	0.3	0.5	1.6

The overall numbers of false positives (FP) and false negatives (FN) drop significantly when using more than one algorithm, as Tables V and VI show. For the FP and FN analysis, we call ENSEMBLE$_{max}$ the number of examples wrongly classified by all three algorithms trained on sets of five features and the one trained on the "combined" feature set. This is the maximum an ensemble composed of the four algorithms could achieve if the combination of their individual results was optimal; here calculated by taking the intersection of the set of examples misclassified by each algorithm. The experiment was run ten times for each attack class to ensure accuracy of the results and to find the attack types in each class that ENSEMBLE$_{max}$ misclassified most. Hence the values displayed in the table are average values over the ten validation sets of the 10-fold cross-validation.

All types of Probe attacks appear at least once as a false negative, however, 'satan' and 'portsweep' seem to be the most difficult attacks to detect. When comparing the problematic instances of 'satan', 'portsweep' and 'ipsweep' with true instances of the same attack types, it seems that src_bytes is the feature that gives the classifiers most trouble. In fact, for probe attacks, src_bytes should be very small, although not equal to zero (since there is always a number of bytes contained in the header); when an example of these attacks has a high value for src_bytes, it goes undetected. This is a big problem since an attacker could easily fill the packets of the attack with random bytes to evade the IDS. It could seem like a good idea to get rid of this feature; however, src_bytes is very important to detect Probe attacks: the only classifier that performs poorly is the one trained on the features selected by LGP, a feature set that does not include src_bytes.

For the U2R class, in general either one false positive or one false negative appears in each test run. The false positive can be explained by the small number of examples in the dataset, only 52 Normal examples are present. The false negative is always a 'rootkit' attack that is wrongly classified as normal traffic, but it is not always the same instance, indicating that some information is missing for the decision tree to classify 'rootkit' attacks. These can be any kind of malware such as worm, Trojan or virus with the ability to hide its presence and actions to the users and processes of a computer; this is called a stealth attack. The diversity found in malware probably has a huge impact on the problem. Moreover, as shown in Table II, there are only 10 'rootkit' attacks in the dataset, increasing the difficulty. Examining the values of these examples for the 14 features of the combined algorithm revealed that almost all 10 instances have very different values for those features. The ENSEMBLE$_{max}$ performs perfectly in most cases, but it is difficult to conclude anything with such a small dataset: One false positive or false negative out of ten instances of the cross-validation set is quite a bad score.

The combination of all algorithms helps to bring down the number of false positives and false negatives also for R2L, but these numbers are again too high for a real-world application. There are eight different types of R2L attacks represented in the training set. After running the experiments ten times, only three types of these attacks trigger false negatives for the ENSEMBLE$_{max}$: 'spy', 'imap', and 'phf'. There is not much documentation about 'spy' attacks, which are not even represented in the test set. However, the signatures of 'imap' and 'phf' are described in [37]. Detection of these attacks requires very specific features. In the case of a 'phf' attack, the IDS according to Kendall (2007) "must monitor http requests watching for invocations of the phf command with arguments that specify commands to be run" [38]. None of the 41 features in the KDD99 dataset gives any information about a specific command being run on the system. It would be impractical to do so for each specific command vulnerable to an attack. However, this could be the reason why the machine learning algorithms are incapable of detecting these kinds of attacks with certainty. Without meaningful information, the algorithms are powerless in building a proper model.

There are two ways to solve this problem, either new features could be added to the dataset or an IDS using signatures of attacks should perform the detection for these particular types of attacks. In the former case, the new features should not be too specific to ensure that new attacks could also be identified. In the second case, the IDS loses its ability to detect similar attacks but its accuracy increases. To detect an 'imap' attack, an IDS should be "programmed to monitor network traffic for oversized Imap authentication strings" [38]. This seems more within reach of our IDS, since service and src_bytes are both represented in the feature set.

ENSEMBLE$_{max}$ was highly successful on the DoS class, returning zero false positives. Table VI shows that the number of false negatives is reduced as well. Three types of attacks trigger false negatives: 'smurf', 'neptune', and 'back'. The first two rarely appear in the list; however, the third seems

Table VII
ACCURACY OF THE MODEL ASSESSMENT

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	93.09	90.00	50.00	79.34
5 SVM features	77.63	40.00	50.00	87.70
5 LGP features	87.48	83.57	61.03	76.10
5 MARS features	84.04	85.00	50.00	82.20
11–14 combined features	79.97	94.29	50.00	85.36

Table VIII
MODEL ASSESSMENT: FALSE POSITIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	86.0	3.0	0.0	69.0
ENSEMBLE$_{\max}$	11.4	1.6	1.0	16.6

Table IX
MODEL ASSESSMENT: FALSE NEGATIVES

Decision Tree Classifier	Probe	U2R	R2L	DoS
41 features	490	11	16,347	7,268
ENSEMBLE$_{\max}$	524	1	7,779	688

to be the most difficult type to handle. This is not a surprise, since to detect a 'back' the IDS must look for a big number of frontslashes ("/") in the request URL [37]. There are no features in the dataset taking this particularity into account. Consequently, the model has to rely on other features to make up for the lack of information, leading to an imperfect result. Nevertheless, as expected, ENSEMBLE$_{\max}$ brings robustness to the accuracy of the IDS.

C. Model Assessment Experiment

In the second round of experiments, several classifiers were trained with different number of features on examples from the training set. Decision tree was again the algorithm used as classifier. The goal of the experiment was to evaluate the model used in the previous experiment on the test set after training on the same number of examples as selected for the training set for each class in the first experiment. As discussed in Section III, the test set is composed of many examples of unseen attacks (attacks that are not represented in the training set). The experiment aimed to assess if the ensemble was capable of generalizing to new types of attacks belonging to the same classes as the ones previously seen.

In most cases, the accuracy of all algorithms degraded drastically in comparison to the first experiment as shown in Table VII, where the values represent one run of the program. In particular, the set of features selected by SVM obtains the worst results, and does not seem to generalize well to new types of attacks. The set selected by LGP managed to keep a respectable accuracy on the Probe class, while all classifiers except SVM showed results very similar to those in the feature selection experiments on U2R, with the "combined" set of features being the best one, outperforming even the algorithm trained with all 41 features in the same way that was observed in the feature selection experiment.

Particularly bad results could be expected for R2L because of the poor distribution of attacks in the training set, and Table VII confirms this: the accuracy of all algorithms is equal or close to the 50 % guessing baseline. Most of the attacks are 'warezclient' (1020 out of 1126 in total for the R2L training set) leaving only 106 instances of all other attack types (seven different types) to train on — and 'warezclient' is not even represented in the test set. There is no chance that the models built would perform well on new attacks (or even on old) with this limited training set. In addition, the results for DoS were much worse than in the first experiment, with the set of features selected by LGP obtaining by far the worst results. Nevertheless, all other algorithms performed better than the one trained with all features.

As Tables VIII and IX show, the ENSEMBLE$_{\max}$ is able to handle part of the new attacks, but does not recognize them as easily as the old ones, and the number of false negatives is very high for most classes. For Probe, the most surprising fact is that the attack 'ipsweep' seems to go undetected almost all the time. This result is unusual because 'ipsweep' was available in the training set and did not cause any trouble in the previous experiment. One reason for this could be if the examples of 'ipsweep' from the test set were very different from the ones in the training set. However, after examining the training set carefully, typical values for the features of an 'ipsweep' attack were observed, and it appears that the values of 'ipsweep' in the test set are in the same range as those in the training set. Overall, the results are not bad. First, almost all old attacks are perfectly detected, especially 'portsweep' and 'satan' which triggered false negatives in the first experiment are now absent from the attacks triggering false negatives. The new attacks are detected most of the time, but the number of false negatives is still too high to be useful in a real-world application. Finally, solving the problem of 'ipsweep' would substantially bring down the number of false negatives.

For U2R, ENSEMBLE$_{\max}$ brings down the average number of false positives to 1.6 and the average number of false negatives to 1.0, respectively, over five runs of the program. As expected, a 'rootkit' attack sometimes goes undetected, just as in the previous experiment. 'ps' also occasionally appears as a false negative. The most surprising result comes from undetected 'buffer overflow', which did not occur in the feature selection experiment. However, 'xterm' and 'sqlattack' are detected all the time, which is good because it means that ENSEMBLE$_{\max}$ generalizes well for the U2R class.

Table X
FEATURE SELECTION ASSESSMENT WITH SVM CLASSIFIER

SVM Classifier	Probe	U2R	R2L	DoS
41 features	99.61	88.00	97.60	99.83
5 SVM features	97.46	67.00	75.11	89.55
5 LGP features	96.91	82.00	65.02	98.83
5 MARS features	86.92	95.00	92.53	97.74
11–14 combined features	99.28	90.00	98.13	99.90

(a) Accuracy

SVM Classifier	Probe	U2R	R2L	DoS
41 features	19.9	15.9	6.7	0.2
ENSEMBLE$_{\max}$	51.9	0.0	2.6	2.2

(b) False positives

SVM Classifier	Probe	U2R	R2L	DoS
41 features	66.4	0.5	50.4	43.6
ENSEMBLE$_{\max}$	18.5	0.1	6.8	9.4

(c) False negatives

Table XI
MODEL ASSESSMENT WITH SVM CLASSIFIER

SVM Classifier	Probe	U2R	R2L	DoS
41 features	50.00	50.00	50.00	50.00
5 SVM features	55.45	50.00	50.15	76.42
5 LGP features	64.45	50.00	51.65	77.23
5 MARS features	84.82	72.86	50.00	76.78
11–14 combined features	51.90	50.00	49.97	72.27

(a) Accuracy

SVM Classifier	Probe	U2R	R2L	DoS
41 features	0.0	70.0	0.0	0.0
ENSEMBLE$_{\max}$	0.0	1.0	0.0	173.4

(b) False positives

SVM Classifier	Probe	U2R	R2L	DoS
41 features	4166.0	0.0	16,347.0	17,761.0
ENSEMBLE$_{\max}$	871.8	0.8	4380.8	715.0

(c) False negatives

The number of false negatives for the R2L class explodes. Old and new types of attacks are similarly misclassified. The only conclusion that can be drawn is that the R2L training set contains too few examples of each type of attack to be of any help.

For DoS, the majority of the false negatives are due to new attacks. Of the old attack types, 'pod' is the only one that regularly triggers a few false negatives, for each run of the program. Other old attacks, such as 'smurf' and 'neptune', sometimes trigger false negatives, but this happens extremely rarely. New attacks are more problematic, with 'mailbomb', 'apache2', 'processtable', and 'udpstorm' recurrently triggering false negatives, even if most of these attacks are detected in general. Even though its generalization power is limited, ENSEMBLE$_{\max}$ performed quite well overall on unseen DoS attacks and helped bring down both false positives and false negatives. This is quite an improvement, but again not enough for a real-world application.

D. Ensembles with Support Vector Machine Classifiers

As displayed in Figure 1, the main algorithms used as classifiers in the ensemble were decision trees (DT). Attempts were also made to use support vector machines (SVM) with a Gaussian radial basis function kernel (which is one of the most powerful machine learning algorithms currently available). However, the results were not very encouraging, as displayed in Table X.

Utilizing support vector machines with Gaussian radial basis functions as the main ensemble classification algorithm for the model assessment experiment yielded even worse results, as shown in Table XI, where the accuracy for many classes and classifiers is around the $50\,\%$ guessing baseline.

One reason why the SVMs do not give accurate results might be that many data points corresponding to different attacks are located among the data points corresponding to normal traffic in the dimensional space. Hence, the SVMs are not able to find hyperplanes that clearly separate the examples in the dataset as attacks or normal traffic. On the other hand, decision trees are able to identify key features that appear at the top of the tree and are in this way able to accurately separate the examples.

Intuitively, it is reasonable to assume that, in this case, using SVMs as multiclass classifiers (one class for each type of attack) instead of as binary classifiers (attack or normal) would give more accurate results, since there should be even more similarities between instances belonging to the same attack type within an attack class ('smurf', 'neptune', etc.) than instances belonging to the same class (DoS, R2L, etc.).

Experiments were also carried out on SVMs using different kernel functions (linear, quadratic and polynomial of order 3). The maximum number of iterations then had to be increased to $100,000$ in order to have the algorithm converge all the time. These experiments were run on the test set only, but several times for each kernel.

Surprisingly, quite good results were obtained on the test set with the linear kernel (except for the R2L class, but that was to be expected). The results were non-conclusive, but sometimes even better than with decision trees. The quadratic kernel also gave some interesting results, but quite limited. Hence, an accuracy of $94\,\%$ for the LGP feature set on Probe attacks could be reached several times. However, the order 3 polynomial kernel produced terrible results across the board, and the algorithm did not converge for the polynomial kernel even after $100,000$ iterations on the feature selection experiment with cross-validation. It transpires that when adding more complex forms for the boundaries, the results are getting worse and worse. The SVMs probably overfit the data in the training phase. This would explain why the results are so bad when increasing the boundary complexity.

Furthermore, SVM was much slower than DT, roughly two orders of magnitude both for training and for classification. In particular classification time is an important criterion to take into account when building a real-world application: if the classifier is very accurate but need too much time to analyse each packet on a network, chances are that many packets will go through without being analysed, leading to poor performance of the intrusion detection system.

VI. Discussion and Future Work

The Feature Selection Assessment experiments (Section V-B) showed that the ensemble approach is indeed a very powerful paradigm that can be used to bring down the number of false positives and false negatives. The lower accuracy observed by individual algorithms is countered by the union of their results. Even with sets containing only five features, the results are very encouraging. Moreover, treating each class of attack as a different problem solved by a specialised algorithm seems to work well when compared to strategies using one algorithm to detect all classes of attacks. "Divide and conquer" and "Unity is strength" seem to be opposite views, but they are actually both applied in this work with impressive results. In general, algorithms using fewer features have slightly lower accuracy and prediction time, but much lower training time.

The results obtained by Mukkamala *et al.* [23] seem to be correct. However, the features selected by LGP give the worst result in most cases except for DoS where it is the feature set selected by SVM that performs poorly. Consequently, the sets of features selected by LGP should be reconsidered for all classes except DoS, while the set of features selected for DoS by SVM should be replaced. The number of different types of attacks that go undetected is very small and only few examples of these attacks are problematic. Most of the time, the problem lays in the lack of information contained in the dataset. Some attacks require very specific features and should probably be handled by specialized programs or signature-based IDSs. The class Probe is a bigger problem since most of the attacks belonging to this class exploit a legitimate feature used by network administrators; as a result, all types of Probe attacks trigger FN at some point, even though 'portsweep' and 'satan' are the most problematic.

A smaller feature set means that less information must be extracted from a network packet in the data preprocessing phase. Since the accuracy is not lowered too much in the best cases, this is a huge improvement that could be used in real IDSs. Moreover, the union of all algorithms using fewer features tremendously improves the accuracy: on average over ten runs of the program, only 0.7 FP and 3.0 FN were observed for Probe over $20,000$ examples, 6.6 FP and 0.5 FN for R2L over $2,252$ examples, 0.3 FP and 0.3 FN for U2R over 104 examples, and 0.0 FP and 1.6 FN for DoS over $20,000$ examples. Even though these results are much better than what could be achieved with a single algorithm, they are still quite far from being useful in a real-world application where the false positives and negatives should be < 1 for some 15 million instances in a 10Gb/s Ethernet network. Arguably, over $90\,\%$ of those instances will be normal traffic containing no attack at all, but $\text{ENSEMBLE}_{\max}$ still has to be improved to stand a chance against clever hackers.

The results described above are the best that an ensemble composed of these algorithms and sets of features could achieve. In its current state, there is no point in building an experiment to assess a real combination of the results of the individual algorithms in the $\text{ENSEMBLE}_{\max}$. Further work will have to be carried out to find the best suitable algorithms and sets of features. Nevertheless, it is interesting to see how well this $\text{ENSEMBLE}_{\max}$ can perform when predicting previously unseen attack types. That was the topic of the second round of experiments, on Model Assessment (Section V-C). Even if $\text{ENSEMBLE}_{\max}$ in general helps tremendously in bringing down the numbers of false positives and false negatives, it is still far from reaching the accuracy appropriate for a real-world application. In particular, datasets that are not carefully designed are proven to be useless in building accurate models of the attacks. This is the case with the R2L training set, which mainly contains examples of the 'warezclient' attack (which is not even represented in the test set) and very few examples of all other types of attacks. The performance of $\text{ENSEMBLE}_{\max}$ was acceptable for the classes of attacks U2R and DoS. The performance on the Probe class was also standard, even though 'ipsweep' attacks went undetected for unknown reasons. Overall, the results of this second round of experiments were not very satisfying, but once again proved the usefulness of the ensemble approach.

In the future, particular attention has to be paid to the features relevant to each attack. New features carrying meaningful information about the attacks must be designed to help the machine learning algorithms to successfully classify all types of attack. The DoS and Probe classes are mostly characterized by time-related features, whereas R2L and U2R mostly are characterized by content-related features extracted from the payload of the network packets.

VII. Conclusions

The aim of this work was to show that ensemble approaches fed with appropriate features sets can help tremendously in reducing both the number of false positives and false negatives. In particular, our work showed that the sets of relevant features are different for each class of attacks, which is why it is important to treat those classes separately. We developed our own IDS to evaluate the relevance of the sets of features selected by Mukkamala *et al.* [4]. This system is an ensemble of four ensembles of decision trees. Each of the four ensembles is in charge of detecting one class of attacks and composed of four decision trees trained on different sets of features. The first three decision trees were fed with sets of five features selected in [4]. The last decision tree was fed with the union of these three sets of five features from which the redundant features were removed.

The experiments showed that these sets were appropriate in most cases. In the first experiment, the set of features selected by linear genetic programming gave the worst results, except for the class `DoS` for which the set of features selected by SVM performed poorly. The second experiment gave less interesting results because of the inappropriate distribution of examples between the training and test sets of the KDD99 data. In particular, the ensemble could not generalize well enough to be useful for the `R2L` class because the training set mainly contains a type of attack that is not represented in the test set. In both experiments, we looked at the number of instances that were misclassified by all four algorithms in order to obtain a result from the best combination of these algorithms. Further work would be required to develop an appropriate decision function combining the results of the different algorithms. However, since the accuracy obtained here was not good enough for a real-world application, designing decision functions was unnecessary. Nevertheless, we are convinced that this work is heading in the right direction in order to overcome the limitations of current intrusion detection systems.

Finally, a thorough analysis of the examples that were misclassified by the ensemble was performed, in particular highlighting the types of attacks that were systematically misclassified by the ensemble. By looking at the signatures of these attacks, we were able to find the reasons for the classification errors. In most cases, the attacks displayed very specific features not captured by the set of variables in the dataset. These attacks should probably be handled by a specialized system or new variables should be developed to train the classifiers.

Acknowledgements

The authors would like to express their thanks to Lillian Røstad (Norwegian University of Science and Technology), Esteban Zimanyi, Olivier Markowitch, Liran Lerman (all at Université Libre de Bruxelles), and the anonymous reviewers for valuable comments on previous versions of the text.

References

[1] A. Balon-Perin, B. Gambäck, and L. Røstad, "Intrusion detection using ensembles," in *Proceedings of the 7th International Conference on Software Engineering Advances*, Lisbon, Portugal, Nov. 2012, pp. 656–663.

[2] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, Jun. 2010, pp. 305–316.

[3] C. Elkan, "KDD Cup 1999: Computer network intrusion detection," Webpage (last accessed: June 12, 2013), 1999, http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection.

[4] S. Mukkamala, A. Sung, and A. Abraham, "Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools," in *Enhancing Computer Security with Smart Technology*, V. R. Vemuri and V. S. H. Rao, Eds. Boca Raton, Florida: CRC Press, Nov. 2005, pp. 125–161.

[5] S. Mukkamala and A. H. Sung, "Identifying significant features for network forensic analysis using artificial intelligent techniques," *International Journal of Digital Evidence*, vol. 1, no. 4, pp. 1–17, 2003.

[6] A. H. Sung and S. Mukkamala, "The feature selection and intrusion detection problems," in *Proceedings of the 9th Asian Conference on Advances in Computer Science*. Chiang Mai, Thailand: Springer-Verlag, May 2004, pp. 468–482.

[7] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?" in *Proceedings of the 13th International Conference on Image Analysis and Processing*. Cagliari, Italy: Springer, Jun. 2005, pp. 50–57.

[8] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, Oct. 1990.

[9] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, Jul. 1990.

[10] J. J. Davis and A. J. Clark, "Data preprocessing for anomaly based network intrusion detection: A review," *Computers & Security*, vol. 30, no. 6-7, pp. 353–375, 2011.

[11] R. Lippmann, "1998 DARPA intrusion detection evaluation data set," Webpage (last accessed: June 12, 2013), 1998, http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/1998data.html.

[12] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, no. 3, pp. 357–374, May 2012.

[13] M. Tavallaee, N. Stakhanova, and A. A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," *IEEE Transactions on Systems, Man, and Cybernetics — Part C: Applications and Reviews*, vol. 40, no. 5, pp. 516–524, Sep. 2010.

[14] D. M. Gregg, W. J. Blackert, D. C. Furnanage, and D. V. Heinbuch, "Denial of service (DOS) attack assessment analysis report," Johns Hopkins University, Baltimore, Maryland, Tech. Rep. AFRL-IF-RS-TR-2001-223, Oct. 2001.

[15] F. Lau and S. H. Rubin, "Distributed denial of service attacks," in *Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics*. Nashville, Tennessee: IEEE, Oct. 2000, pp. 2275–2280.

[16] C. Patrikakis, M. Masikos, and O. Zouraraki, "Distributed denial of service attacks," *The Internet Protocol Journal*, vol. 7, no. 4, pp. 13–35, Dec. 2004.

[17] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole, "Buffer overflows: attacks and defenses for the vulnerability of the decade," in *Foundations of Intrusion Tolerant Systems (Organically Assured and Survivable Information Systems 2003)*, J. H. Lala, Ed. Los Alamitos, California: IEEE Computer Society, Jan. 2003, pp. 227–237.

[18] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2nd International Conference on Computational Intelligence for Security and Defense Applications*. Ottawa, Ontario, Canada: IEEE, Jun. 2009, pp. 53–58.

[19] J. P. Anderson, "Computer security threat monitoring and surveillance," James P. Anderson Co., Fort Washington, Pennsylvania, Tech. Rep., Apr. 1980.

[20] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, Feb. 1987.

[21] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A network security monitor," in *Proceedings of the IEEE Symposium on Security and Privacy*. Los Alamitos, California: IEEE Computer Society, May 1990, pp. 296–304.

[22] S. X. Wu and W. Banzhaf, "The use of computational intelligence in intrusion detection systems: A review," *Applied Soft Computing*, vol. 10, no. 1, pp. 1–35, Jan. 2010.

[23] S. Mukkamala, A. H. Sung, and A. Abraham, "Intrusion detection using an ensemble of intelligent paradigms," *Journal of Network and Computer Applications*, vol. 28, no. 2, pp. 167–182, Apr. 2005.

[24] S. Chebrolu, A. Abraham, and J. P. Thomas, "Hybrid feature selection for modeling intrusion detection systems," in *Proceedings of the 11th International Conference on Neural Information Processing*, ser. Lecture Notes in Computer Science, N. R. Pal, N. Kasabov, R. K. Mudi, S. Pal, and S. K. Parui, Eds., vol. 3316. Calcutta, India: Springer, Nov. 2004, pp. 1020–1025.

[25] ——, "Feature deduction and ensemble design of intrusion detection systems," *Computers & Security*, vol. 24, no. 4, pp. 295–307, Jun. 2005.

[26] A. Abraham and J. P. Thomas, "Distributed intrusion detection systems: A computational intelligence approach," in *Information Security and Ethics: Concepts, Methodologies, Tools, and Applications*, H. Nemati, Ed. Hershey, Pennsylvania: IGI Global, Sep. 2005, vol. 5, pp. 105–135.

[27] S. Peddabachigari, A. Abraham, C. Grosan, and J. P. Thomas, "Modeling intrusion detection system using hybrid intelligent systems," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 114–132, 2007.

[28] A. Abraham and R. Jain, "Soft computing models for network intrusion detection systems," in *Classification and Clustering for Knowledge Discovery*, ser. Studies in Computational Intelligence, S. K. Halgamuge and L. Wang, Eds. Berlin Heidelberg, Germany: Springer, Oct. 2005, vol. 4, pp. 191–207.

[29] A. Abraham, R. Jain, J. P. Thomas, and S.-Y. Han, "D-SCIDS: Distributed soft computing intrusion detection system," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 81–98, Jan. 2007.

[30] A. Zainal, M. A. Maarof, S. M. Shamsuddin, and A. Abraham, "Ensemble of one-class classifiers for network intrusion detection system," in *Proceedings of the Fourth International Conference on Information Assurance and Security*. Naples, Italy: IEEE Computer Society, Sep. 2008, pp. 180–185.

[31] G. Folino, C. Pizzuti, and G. Spezzano, "GP ensemble for distributed intrusion detection systems," in *Proceedings of the 3rd International Conference on Advances in Pattern Recognition*, Bath, England, Aug. 2005, pp. 54–62.

[32] ——, "An ensemble-based evolutionary framework for coping with distributed intrusion detection," *Genetic Programming and Evolvable Machines*, vol. 11, no. 2, pp. 131–146, Jun. 2010.

[33] E. Bahri, N. Harbi, and H. N. Huu, "Approach based ensemble methods for better and faster intrusion detection," in *Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems*, ser. Lecture Notes in Computer Science. Torremolinos-Málaga, Spain: Springer, Jun. 2011, pp. 17–24.

[34] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[35] P. Zhang, X. Zhu, Y. Shi, L. Guo, and X. Wu, "Robust ensemble learning for mining noisy data streams," *Decision Support Systems*, vol. 50, no. 2, pp. 469–479, Jan. 2011.

[36] H. Gunes Kayacik, A. Nur Zincir-Heywood, and M. I. Heywood, "A hierarchical SOM-based intrusion detection system," *Engineering of Applied Artificial Intelligence*, vol. 20, no. 4, pp. 439–451, Jun. 2007.

[37] K. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems," Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, Jun. 1999.

[38] ——, "Intrusion detection attacks database," Webpage (last accessed: June 12, 2013), 2007, http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/docs/attackDB.html.