POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System*

Damiano Bolzoni, Sandro Etalle, Pieter Hartel
University of Twente,
Distributed and Embedded System Group,
P.O. Box 2100, 7500 AE Enschede, The Netherlands
{*damiano.bolzoni, sandro.etalle, pieter.hartel*}*@utwente.nl*

Emmanuele Zambon
Universita' Ca'Foscari di Venezia,
Dipartimento di Informatica,
Via Torino 155, 30172 Mestre (VE), Italy
ezambon@dsi.unive.it

Abstract

We present POSEIDON, a new anomaly-based network intrusion detection system. POSEIDON is payload-based, and has a two-tier architecture: the first stage consists of a Self-Organizing Map, while the second one is a modified PAYL system. Our benchmarks on the 1999 DARPA data set show a higher detection rate and lower number of false positives than PAYL and PHAD.

1 Introduction

Intrusion detection systems were introduced by Anderson [1] and formalized later by Denning [11]. Nowadays, there exist two main types of network intrusion detection methods: *anomaly-based* and *signature-based*. In signature-based methods, (e.g. Snort [29, 30]) a characteristic trait of the intrusion is developed off-line, and then loaded in the intrusion database before the system can begin to detect this particular intrusion. This usually yields good results in terms of low false positives, but has drawbacks: firstly in most systems, *all* new attacks will go unnoticed until the system is updated, creating a window of opportunity for attackers to gain control of the system under attack. Secondly, only known attacks can be detected, and while this could be acceptable for detecting attacks to e.g., the OS, it makes it much harder to use signature-based system for protecting web-based services, because of their ad-hoc nature. Notably, the protection of web-services is becoming a high-impact problem [15].

Anomaly-based systems (ABS), on the other hand, build statistical models that describe the normal behaviour of the network, and flag any behaviour that significantly deviates from the norm as an attack. This has the advantage that new attacks will be detected as soon as they take place. ABS can be applied also to ad-hoc networked systems such as web-based services. The disadvantage is that ABS needs an extensive model building phase: a significant amount of data (and thus a significant period of time) is needed to build accurate models of legal behaviour.

Most network intrusion detections systems in use today are signature-based, however, new attacks are devised with increasing frequency every day (see [15] for weekly and monthly single attack rates), so anomaly-based systems become increasingly attractive.

Every network intrusion detection system suffers from (1) false positives (false alarms), in which legal behaviour is incorrectly flagged as an attack and (2) false negatives, or misses, in which true attacks are undetected. Anomaly-based systems are more vulnerable to these problems than signature-based systems because they use statistical models to detect intrusions.

ABS can extract information to detect attacks from different layers: packet headers, packet payload or both. *Header information* is mainly useful to recognize attacks aiming at vulnerabilities of the network stack implementation or probing the operating system to identify active network services. On the other hand, *payload information* is most useful to identify attacks against vulnerable applications (since the connection that carries the attack is estab-

*This research is supported by the research program Sentinels (http://www.sentinels.nl). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

Proceedings of the Fourth IEEE International Workshop on Information Assurance (IWIA'06)
0-7695-2564-4/06 $20.00 © 2006 IEEE

lished in a normal way) [32]. Without pretending to be globally better than other types of ABS, payload-based systems have importance of their own, as they are particularly suitable for detecting popular attacks such as those on the HTTP protocol, and worms (see Wang and Stolfo [31] and Costa et al. [9] for a discussion). Notably, PAYL and the system of Kruegel et al. [19] are mainly payload-based, while PHAD [24] is partly payload based.

Contribution In this paper we propose POSEIDON (Payl Over Som for Intrusion DetectiON): a two-tier network intrusion detection architecture. The first tier consists of a self-organizing map (SOM), and is used exclusively to classify payload data; the second tier consists of a slight modification of the well-known PAYL system [32] (see Figure 1).

POSEIDON is payload-based: it uses only destination address and service port numbers to build a profile for each port monitored, and it does *not* consider other header features.

We have extensively benchmarked our system w.r.t. PAYL [32] (also by replicating the PAYL experiments) and PHAD [24] using the 1999 DARPA benchmark [23]. PAYL and PHAD are the reference ABS based on payload analysis. On this data set, our experiments show:

- a *higher* detection rate and *lower* number of false positives than PAYL and PHAD.
- a reduction of the number of profiles used w.r.t. PAYL. This has a positive influence on the runtime efficiency of the system.

Incidentally, being payload-based, our system takes into consideration only what Mahoney and Chan [25] call the *legitimate* data of the 1999 DARPA data set, implying that we can legitimately expect that the system in real life performs as well as it does on the DARPA benchmark.

Let us now explain the reasons that brought us to the development of this architecture. First of all, for the classification phase, we believe that a self-organizing map - in general - can yield to a high quality classification, i.e. clusters with a high intra-cluster similarity and high inter-cluster dissimilarity, without having to take into account the length of the packet. This can be used to build good profiles.

At the same time, we believe that a SOM is not as effective when it comes to the detection phase, i.e. to finding whether a given packet is anomalous w.r.t. the cluster it has been classified in. In a SOM, the detection phase is accomplished by comparing the current packet quantization error with matching cluster quantization error: this method can be heavily influenced by payload byte order, because it is based on a distance function. For the detection, we believe that the n-gram algorithm used by PAYL is more suitable.

On the other hand, we believe that the Achilles' heel of the PAYL architecture lies in the classification it adopts: the algorithm uses packet payload length information to classify packets and thus to define clusters. This, together with the fact that - for efficiency reasons - clusters have to be merged, yields in our opinion to a too low *intra-cluster* similarity: two packets belonging to the same cluster can present very different byte distribution, without that this indicates an attack.

By combining a SOM with the n-gram algorithm we obtained an architecture that combines the advantages of the SOM (the realization of clusters with high intra-cluster similarity) with those of PAYL (the ability to detect when a packet is anomalous w.r.t. a given cluster). The results we have obtained on the DARPA substantiate our beliefs.

This paper is structured as follows: Section 2 presents the internals of POSEIDON and of PAYL; in Section 3 we describe benchmarking experiments and compare obtained results with PAYL and PHAD. In Section 4 we discuss other related work. Finally, in Section 5 we draw our conclusions and set the course for further developments. In the appendix we report the pseudo-code of POSEIDON.

2 Architecture

Network intrusion detection systems are either *packet-oriented* or *connection-oriented*. In the former architecture, every packet is analysed as soon as it arrives, without trying to correlate it with previous collected data. On the other hand, connection-oriented systems work either by (a) reassembling the whole connection (commonly only from client to server) - waiting until the connection is closed - to analyse the connection payload, or (b) by gathering statistics which consider, e.g., the amount of bytes transmitted and received, the duration of the connection, the protocol type and final connection status.

POSEIDON, like most network intrusion detection systems, is packet-oriented. This architecture presents two main advantages: firstly, POSEIDON can identify and block an attack *while* it is taking place (intrusion prevention). Secondly, connection-based systems are computationally more expensive, in particular they require a huge amount of memory resources to keep all the segments to analyse. This makes connection-based system more suitable for off-line analysis. On the other hand, connection-based systems support a finer-grained analysis.

Our starting point is the PAYL architecture. Our algorithm receives as input a packet and *classifies* the packet, without prejudice for any of its properties, such as length, destination port or application data semantics. The idea is that the classifier keeps as much information as possible

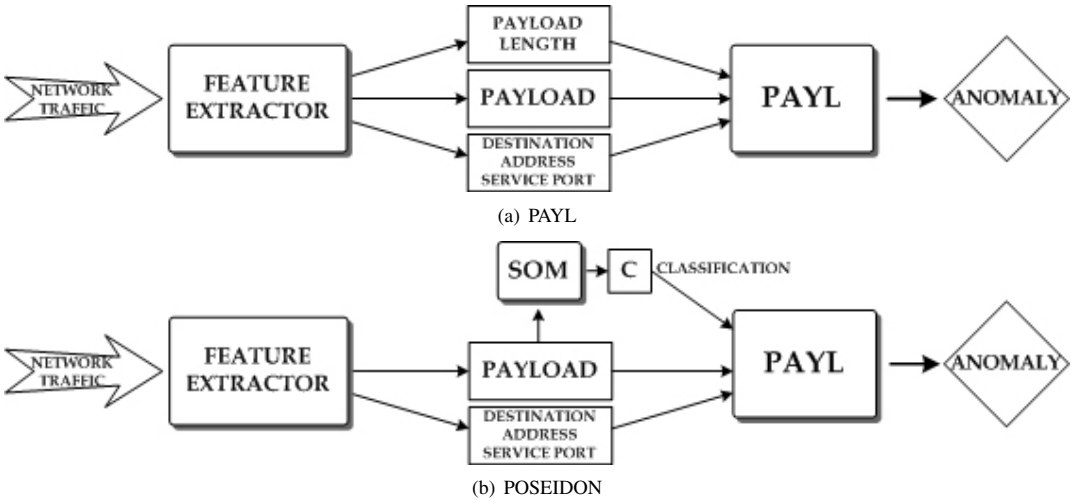

Figure 1. PAYL and POSEIDON architectures

about packets (e.g. high-dimensional data) for the anomaly detection phase: we also want the classifier to operate in an unsupervised manner. This is a typical clustering problem which can be properly tackled using neural networks in general and Self-Organizing Maps (SOM) [18] in particular. SOMs have been widely used in the past both to classify network data and to find anomalies. Here, we use them for pre-processing.

Our architecture combines a SOM with a modified PAYL algorithm. Figure 1 shows a comparison between our architecture and PAYL's.

We now give a high-level description of the algorithms underlying our system, a more formal description is reported in the appendix. We first describe the SOM. Later in the section, we introduce PAYL, focusing on the main differences between our approach and the PAYL approach towards classification of network data.

2.1 SOM classification model

Self organizing maps are defined as topology-preserving single-layer maps in which the topological structure, imposed on the nodes in the network, is not changed during classification (preserving neighbourhood relations) and there is only one layer of nodes. A SOM is suitable to analyse high-dimensional data and belongs to the category of competitive learning networks [18]. Nodes are also called *neurons*, to remind us of the artificial intelligence nature of the algorithm. Each neuron n has a *vector of weights* w_n associated to it: the dimension of the weights arrays is equal to the length of longest input data. These arrays (also referred as *reference vectors*) determine the SOM behaviour.

To accomplish the classification, SOM goes through three phases: initialization, training and classification.

Initialization First of all, some system parameters (number of nodes, learning rate and radius) have to be fixed by e.g. the IDS technician. The number of nodes directly determines the classification given by the SOM: a small network will classify different data inputs in the same node while a large network will produce a too sparse classification. Afterwards, the array of node weights is initialized, usually with random values (in the same range of input values).

Training The training phase consists of a number of iterations (also called *epochs*). At each iteration one input vector x is processed as follows: x is compared to all neuron weight arrays w_n with a distance function (Euclidean or Manhattan): the most similar node (also called *best matching unit*, BMU) is then identified.

After the BMU has been found, the neighbouring neurons and the BMU itself are updated. The following update parameters are used: the neighbourhood is governed by the *radius* parameter (r) and the magnitude of the attraction is affected by the *learning rate* (α).

During this phase, the map tends to converge to a stationary distribution, which approximates the probability density function of the high-dimensional input data.

As the learning proceeds and new input vectors are given to the map, the learning rate and radius values gradually decrease to zero.

Classification During the classification phase, the first part of the training phase is repeated for each sample: the input data is compared to all the weight arrays and the most

similar neuron determines the classification of the sample (but weights are not updated). The winning neuron is then returned.

2.2 PAYL classification model

PAYL, is a n-gram [10] analysis algorithm, and uses a classification method based on clustering of packet payload data length.

PAYL classifies packets on the *length of the payload*. During the training phase, for a given training data set, PAYL computes a set of *models* M_{ijk}. For each incoming packet, with destination address j and destination port k and payload length i, M_{ijk} stores incrementally the average byte frequency and the standard deviation of each byte frequency. During the detection phase, the same values are computed for incoming packets and then compared to model values: a significant difference from the norm produces an alert. To compare models, PAYL uses a simplified version of the Mahalanobis distance, which has the advantage of taking into account not only the average value but also its variance and the covariance of the variables measured.

The maximum amount of space required by PAYL is: $p*l*k$, where p is the total number of ports monitored (each host may have different ports), l is the length of the longest payload and k is a constant representing the space required to keep the mean and the variance distribution values for each payload byte (PAYL uses a fixed value of 512).

To reduce the otherwise large number of models to be computed, PAYL organizes models in clusters. After comparing two neighbouring models using the Manhattan distance, if the distance is smaller than a given threshold t, models are merged: the means and variances are updated to produce a new combined distribution. This process is repeated until no more models can be merged. Experiments with PAYL show [32] that a reduction in the number of model of up to a factor of 16 can be achieved.

Modification to PAYL Our modification to PAYL works as follows: we pre-process each packet, using the SOM. Afterwards PAYL uses the class value given by the SOM (*winning neuron*) instead of the payload length. Technically PAYL, instead of using model M_{ijk}, uses the model M_{njk} where j and k are the usual destination address and port and n is the classification derived from the neural network. Then, mean and variance values are computed as usual.

Having added SOM to the system we must allow for both the SOM and PAYL to be trained separately. Regarding resource consumption, we have to revise the required amount of space to: $p * n * k$, where the new parameter n indicates the amount of SOM network nodes.

3 Tuning and Experiments

In this section, we show the results of our benchmarks and compare the performance of POSEIDON with PAYL and PHAD. PAYL and PHAD are the two reference ADS based on payload. They are the only two ABS based on payload which have published their detection rate on the DARPA 1999 data set.

3.1 SOM parameters tuning

The SOM algorithm needs several parameters on start-up: the total number of network nodes, the function used to compute the distance between vectors and the values of the *learning rate* and *update radius*. For the sake of transparency, we report here the values used in our experiments.

Concerning the number of neurons, a small network would yield a too course classification, while a large network will produce a sparse classification. In addition, it is worth bearing in mind that the computational load increases quadratically with the number neurons.

Experimenting with different initialization parameters and using the *quantization error* method [18] to evaluate the classification given by the network, we found the best SOM with the following parameters:

- Number of neurons: 96 (rectangular network of 12 by 8).
- Learning rate: 0.1.
- Update radius: 4.
- Distance function: Manhattan.

Hinneburg et al. [13] state that Manhattan distance performs better than Euclidean distance in presence of high-dimensional data: our experiments substantially confirm this statement also in the case of network data analysis.

3.2 Experiments

We have benchmarked POSEIDON against PAYL (also by replicating the experiment on PAYL) and PHAD, using the same data used by PAYL and PHAD: the DARPA 1999 data set [23]. This standard data set is used as reference by a number of researchers (e.g. [24, 27, 32]), and offers the possibility of comparing the performance of various IDS. This data set has been criticized because of the environment in which data were collected [26]; as explained by Mahoney and Chan [25], it is possible to tune an IDS in such a way that it scores particularly well on this particular data set: some attributes – specifically: remote client address, TTL, TCP options and TCP window size – have a

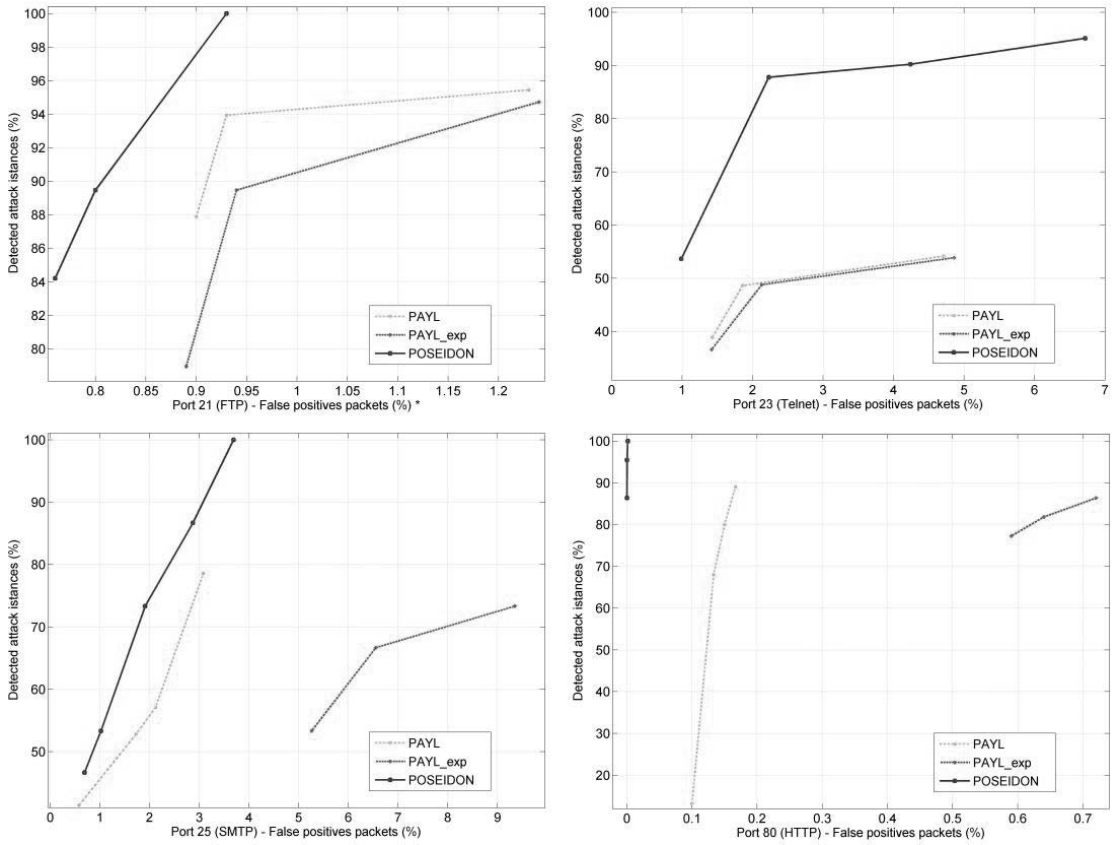

Figure 2. Detection rates for ports 21 (FTP), 23 (Telnet), 25 (SMTP) and 80 (HTTP): the x-axis and y-axis present false positive rate and detection rate respectively. POSEIDON presents always a higher detection rate compared with PAYL at the same false positive rate. For the graph relative to port 21 see Remark in Section 3.2.

small range in the DARPA simulation, but have a large and growing range in real traffic. IDS which take into account the above-mentioned attributes are likely to score much better on the DARPA set than in real life. Since our system does not consider these attributes, we can legitimately expect that the system in real life performs as well as it does on the DARPA benchmark.

To compare our model with PAYL, we apply the same restrictions and conditions used by Wang and Stolfo [32]: we focus only on inbound TCP packets, with data payload, directed to hosts 172.016.0.0/16 and ports 1-1024.

We train the SOM clustering algorithm using internal network traffic of week 1 and week 3 (12 days, 2.444.591 packets, attack free): for each different protocol we use a different SOM. Then, we use the same data to build PAYL models taking advantage of the classification given by the neural network.

After this double training phase, it is possible to use the testing weeks (4 and 5) to benchmark the network intrusion detection algorithm. This data contains several attack instances (97 payload-based attacks are detectable applying the same traffic filter mentioned above), as well as legal traffic, directed against different hosts of the internal network: the attack source can be situated both inside and outside the network.

Figure 2 shows a detailed comparison of PAYL and POSEIDON in terms of percentage of true negatives (reported on the y axis) w.r.t. the percentage false positives (x axis). Table 1 reports a summary of these results: the first column reports PAYL's statistics as we have inferred them from the graphs reported by Wang and Stolfo [32]. The second column reports the figures we obtained by repeating Wang and Stolfo's benchmarks. In the repeated PAYL experiments we used an **un-clustered** architecture, which yields on one

		PAYL	PAYL_exp	POSEIDON
Number of profiles used		4065	(11312 - unclustered)	1622
HTTP	DR	89,00%	90,00%	100,00%
	FP	0,17%	0,73%	0,0016%
FTP	DR	95,50%	94,74%	100,00%
	FP	1,23%	11,41% (1,21%*)	11,31% (0,93%*)
Telnet	DR	54,17%	53,65%	95,12%
	FP	4,71%	4,94%	6,72%
SMTP	DR	78,57%	73,34%	100,00%
	FP	3,08%	8,35%	3,69%
Overall DR with FP $< 1\%$		58,8% (57/97)		73,2% (71/97)*

Table 1. Comparison between PAYL, our implementation of PAYL (PAYL_exp) and POSEIDON; DR stands for detection rate, while FP is the false positive rate

Type	Attack	PHAD	POSEIDON
Probe	ntinfoscan	66,67% (2/3)	100% (3/3)
Denial of Service	apache2	100% (3/3)	100% (3/3)
	back	0% (0/4)	100% (4/4)
	crashiis	71,43% (5/7)	100% (7/7)
Remote to Local	phf	66,67% (2/3)	100% (3/3)
	ppmacro	33,34% (1/3)	100% (3/3)
Overall detection rate		65% (13/20)	100% (20/20)

Table 2. Comparison between PHAD and POSEIDON detection rates.

hand to a higher number of profiles, and on the other hand to a different classification. The third column reports POSEIDON's result. Is it possible to observe that POSEIDON overcomes PAYL on every benchmarked protocol: there is a remark about FTP protocol (see the next paragraph).

Remark During FTP protocol benchmarks we found a high rate of false positives (more than 3000 packets) both with PAYL and with POSEIDON: all these packets are sent by the same source host, which is sending FTP commands in a way that is typical of the Telnet protocol (one character per packet, with the TCP flag *PUSH* set). These packets are marked as an attack because the training model does not contain this kind of traffic over the FTP control channel port, although it is normal traffic. During our experiments with PAYL we found the same behaviour: for this reason we decided to present benchmarks results of PAYL and POSEIDON also without taking into account these packets (the figures marked with an asterisk * in Table 1 and the graph in Figure 2).

Table 2 compares our results with PHAD: it is not possible to make a full comparison between the two systems, because of the restrictions used by PHAD authors (they restrict to a maximum total amount of 100 false positives during 10 days of testing). Nonetheless, we could legitimately compare the two systems on the HTTP protocol, on which POSEIDON meets the restrictions above.

Unfortunately, there is no other public available data set suitable to compare our approach with previous researches on anomaly intrusion detection: many authors use the KDD 99 data set [5] in which regrettably payload data is discarded. Because we use payload information, we can not use this data set to benchmark POSEIDON and models that use this data set are not directly comparable with ours.

Concluding, the significant achieved improvement over PAYL is determined by a better distribution of mean and variance value within categories, obtained with introduction of a new classification algorithm (SOM).

4 Related work

Network intrusion detection systems based on anomaly detection have been widely studied for two decades. We recall that anomaly detection systems can operate in various manners, sometimes extracting features from packet headers and sometimes from payload data.

In this section we report on related work. First we describe other neural network-based systems then we address statistical-based systems.

4.1 Neural networks based systems

We start by presenting other neural-network based IDS. We cannot benchmark these systems with POSEIDON because their authors use either private data sets (Cannady [6], Labib and Vemuri [20] and Ramadas et al. [28]), or data sets that do not contain payload information (Depren et al. [12]) or do not provide precise statistics (Nguyen [27]).

Cannady [6] proposes a SOM-based IDS in which network packets are first classified according to nine features and then presented to the neural network. Attack traffic is generated using a security audit tool. The author extends this work in Cannady [7, 8].

Nguyen [27] uses a one-tier architecture, consisting of a SOM, to detect two attacks in the 1999 DARPA data set: the first one (*mailbomb*) against the SMTP service, and the other one (*guessftp*) against FTP.

Labib and Vemuri [20] use a SOM to identify Denial of Service attacks. They discard information about payload and use only packet header information; their data is collected from a private network (described in a general way) and is not publicly available.

Ramadas et al. [28] use a SOM to detect attacks against DNS and HTTP services (using a private data set): they use a pre-processor to summarize some connection parameters (source and destination host and port) and then add several values to track connections behaviour: the information is then merged in a data structure used to fire events related to the connection and to feed the neural network.

Depren et al. [12] present a hybrid IDS based on self-organizing maps and benchmark it on the KDD 99 data set [5]. They feed the neural networks (one for each protocol type) with six features extracted from each connection (duration, protocol type, service type, status, total bytes sent and received) and then use the quantization error method to detect anomalies. The system is connection-oriented, therefore attacks can be detected only when the connection is completely re-assembled. Regarding their architecture, the authors state that the SOM used to model TCP connections uses 1515 neurons; which in our opinion is quite large, if compared with the ones used by our system.

4.2 Statistical-based systems

In addition to ADS based on neural networks, there exist ADS employing statistical models to detect anomalous behaviour. We now report on them. Again, we cannot benchmark them against POSEIDON because they either use only header information (Hoagland [14], Javitz and Valdes [16]) or employ benchmarking data that is not publicly available (Kruegel et al. [19]).

Barbará et al. [3, 2] use data mining techniques to detect attacks on network infrastructures: their system ADAM first applies association rules techniques to identify abnormal events in traffic data; then a classification algorithm is used to classify the abnormal events into normal instances and abnormal instances. The original work has been expanded in [4]. Lee et al. [21, 22] propose a comprehensive framework based on data mining. For a complete overview of data mining techniques applied to intrusion detection see Julisch [17].

The SPADE [14], NIDES [16] and PHAD [24] systems rely on statistical models computed on normal network traffic: they work by extracting features from the packet header fields and trigger an alarm when they recognize a significant deviation from the normal model; most of the features extracted are related to IP addresses (source and destination), destination service port and TCP connection state (PHAD uses up to 34 attributes coming from Ethernet, IP and application layer protocols packets). Our approach differs from the one mentioned here in the following aspects: (a) it is payload-based: we use only destination address and service port numbers to build a profile for each port monitored, without taking care of other header features (of the above systems only PHAD considers payload information, we have compared it with our system in the previous section). (b) We have a two-tier architecture in which the SOM is used only to pre-process information.

Shifting to payload-based systems, Kruegel et al. [19] show that it is possible to find the description of a system that computes a payload byte distribution and combines this information with extracted packet header features: they first sort the resultant ASCII characters by frequency and then aggregate them into six groups. As argued by Wang and Stolfo [32], this leads to a very course classification of the payload.

PAYL works in a way similar to Kruegel et al. [19] but models the full byte distribution based on payload data length and operates a clustering phase to cover possible missing lengths. The PAYL architecture is made up of a single tier, while our architecture has two different layers: the first one, made up by a SOM, is delegated to classify packets only using payload data information, without using payload length value. The second layer is a modified version of PAYL that computes byte distribution models using the classification information coming from the first layer and extracting destination IP address and service port from packets header.

Zanero [34] presents a two-tier payload-based system that combines a self-organizing map with a modified version of SmartSifter [33]. While this architecture is similar to POSEIDON, a full comparison is not possible because the benchmarks in [34] concern only the FTP service an no details are given about experiments execution. A two-tier architecture for intrusion detection is also outlined in Zanero and Savaresi [35].

5 Conclusion

We present an approach to Network Intrusion Detection that involves the combination of two different techniques: a self-organizing map and the PAYL architecture. We modify the original PAYL to take advantage of the unsupervised classification given by the SOM, which then functions as pre-processing stage.

Our experiments on the DARPA set show that our approach reduces the number of profiles used by PAYL (payload length can vary between 0 and 1460 in a Local Area Network, while the SOM neural network used in our experiments has less than one hundred nodes). Our experiments show that PAYL without SOM requires 3 times as many profiles as with the SOM pre-processing (see Table 1).

We benchmark POSEIDON extensively against the PAYL algorithm and data sets showing a higher detection rate and lower false positives rate.

Acknowledgments We thank Herbert Bos for his valuable comments.

A Appendix: POSEIDON inner functions

In this section we describe the inner mathematical functions and algorithms used by POSEIDON.

A.1 SOM algorithm

DATA TYPE

$RR = [0.0..255.0]$
/* Reals (Double) between 0.0 and 255.0 */
$l = \textit{length of the longest packet payload}$
$PAYLOAD = \textit{array}\ [1..l]\ \textit{of}\ [0..255]$

DATA STRUCTURE

$N = \textit{non} - \textit{empty finite set of neurons}$

$\textit{for each}\ n \in N\ \textit{let}$
$w_n := \textit{array}\ [1..l]\ \textit{of}\ RR$
/* array of weights associated */
/* to each neuron n */
$\alpha_0 \in \mathbb{R}$ /* Initial learning rate */
$\alpha := \alpha_0$ /* Current learning rate */
$r_0 \in \mathbb{R}$ /* Initial radius */
$r := r_0$ /* Current radius */
$\tau \in \mathbb{N}$ /* Number of training epochs */
$k \in \mathbb{N}$ /* Smoothing factor */

INIT PHASE

$\textit{for each}\ n \in N$
$\textit{for}\ i := 1\ \textit{to}\ l$
$w_n[i] := \textit{random}(RR)$
/* Initialize with values in RR */

TRAINING PHASE

INPUT:
$x_t : PAYLOAD$

$\textit{for}\ t := 1\ \textit{to}\ \tau$

/* Find winning neuron */
$win\_dist := +\infty$
$win\_neuron := n_0$

$\textit{for each}\ n \in N\ \textit{do}$
$dist := manhattan\_dist(x_t, w_n)$
$\textit{if}\ (dist \leq win\_dist)\ \textit{then}$
$win\_dist := dist$
$win\_neuron := n$
$\textit{end if}$
$done(for)$

/* Process neighbouring neurons */
$N_n = \{n \in \mathbb{N} \mid trig\_dist(n, win\_neuron) \leq r\}$

$\textit{for each}\ n_n \in N_n$
$\textit{for}\ i := 1\ \textit{to}\ l$
$w_{n_n}[i] := w_{n_n}[i] + \alpha * (w_{n_n}[i] - x_t[i])$

$\alpha := \alpha_0 * \frac{k}{k+t}$
$r := r_0 * \frac{\tau - t}{\tau}$

$done(for)$

CLASSIFICATION PHASE

INPUT:
$x : PAYLOAD$

OUTPUT:
$win\_neuron \in \mathbb{N}$

$win\_dist := +\infty$
$dist := win\_dist$
$win\_neuron := n_0$

$\textit{for each}\ n \in N\ \textit{do}$
$dist := manhattan\_dist(x, w_n)$
$\textit{if}\ (dist \leq win\_dist)\ \textit{then}$
$win\_dist := dist$
$win\_neuron := n$
$\textit{end if}$
$done(for)$

return win_neuron

A.2 PAYL algorithm

DATA TYPE

feature vector = *RECORD* [
    mean = *array* [1..256] *of Real*,
        /* average byte frequency */
    stdDev = *array* [1..256] *of Real*
        /* standard deviation of each */
        /* byte frequency */
]
profile = *RECORD* [
    ip $\in \mathbb{N}$, /* destination host address */
    sp $\in \mathbb{N}$, /* destination service port */
    fv = *finite set of n feature vectors*
]
/* for each port monitored a profile */
/* with n feature vectors is associated */

DATA STRUCTURE

P = *set of finite profiles*
threshold $\in \mathbb{R}$
/* numeric value used for anomaly */
/* detection given by user */

TRAINING PHASE

INPUT:
ip : *IP address* $\in \mathbb{N}$
sp : *service port* $\in \mathbb{N}$
n : *SOM classification*
x : *PAYLOAD*

for each p $\in$ *P do*
    if (*p.ip* = *ip and p.sp* = *sp*) *then*
        fv = *p.getFV*(*n*)
        /* get feature vector with index n */
        fv.update(*x*)
        /* update byte frequency distributions */
    end if
done(*for*)

TESTING PHASE

INPUT:
ip : *IP address* $\in \mathbb{N}$
sp : *service port* $\in \mathbb{N}$
n : *SOM classification*
x : *PAYLOAD*

OUTPUT:
isAnomalous : *BOOLEAN*
    /* is the packet anomalous ? */

dist := $+\infty$
isAnomalous := *FALSE*

for each p $\in$ *P do*
    if (*p.ip* = *ip and p.sp* = *sp*) *then*
        fv := *p.getFV*(*n*)
        /* get feature vector with index n */
        dist := *fv.getDistance*(*x*)
        /* get the distance between input */
        /* data and associated profile */
    end if
done(*for*)

if (*dist* $\geq$ *threshold*) *then*
    isAnomalous := *TRUE*
end if

return isAnomalous

References

[1] J. P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P Anderson Co. Fort Washington, PA, April 1980.

[2] D. Barbará, J. Couto, S. Jajodia, and N. Wu. ADAM: a testbed for exploring the use of data mining in intrusion detection. *SIGMOD Record*, 30(4):15–24, 2001.

[3] D. Barbará, J. Couto, S. Jajodia, and N. Wu. ADAM: Detecting Intrusions by Data Mining. In *IAW '01: Proc. 2nd IEEE SMC Information Assurance Workshop*, 2001. URL http://www.itoc.usma.edu/Workshop/2001/Authors/Submitted_Abstracts/paperT1A3(21).pdf.

[4] D. Barbará, J. Couto, S. Jajodia, and N. Wu. Detecting Novel Network Intrusions using Bayes Estimators. In *SIAM '01: Proc. 1st SIAM International Conference on Data Mining*, 2001. URL http://www.siam.org/meetings/sdm01/pdf/sdm01_29.pdf.

[5] S. D. Bay, D. Kibler, M. Pazzani, and P. Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *SIGKDD Exploration: Newsletter of SIGKDD and Data Mining*, 2(2):81–85, 2000.

[6] J. D. Cannady. Artificial neural networks for misuse detection. In *NISSC' 98: Proc. 21st National Information Systems Security Conference*, pages 443–456, 1998.

[7] J. D. Cannady. *An adaptive neural network approach to intrusion detection and response*. PhD thesis, Nova Southeastern University, 2000.

[8] J. D. Cannady. Next Generation Intrusion Detection: Autonomous Reinforcement Learning of Network Attacks. In *NISSC '00: Proc. 23rd National In-*

formation Systems Security Conference, 2000. URL http://csrc.nist.gov/nissc/2000/proceedings/papers/033.pdf.

[9] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: end-to-end containment of Internet worms. In *SOSP '05: Proc. 20th ACM Symposium on Operating Systems Principles*, pages 133–147. ACM Press, 2005.

[10] M. Damashek. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

[11] D. E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[12] M. O. Depren, M. Topallar, E. Anarim, and K. Ciliz. Network Based Anomaly Intrusion Detection using Self Organizing Maps (SOMs). In *SIU '04: Proc. 12th IEEE National Conference on Signal Processing and Applications*, pages 76–79, 2004.

[13] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What Is the Nearest Neighbor in High Dimensional Spaces? In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, editors, *VLDB '00: Proc. 26th International Conference on Very Large Data Bases*, pages 506–515. Morgan Kaufmann, 2000.

[14] J. Hoagland. Stealthy Portscan & Intrusion Correlation Engine (SPADE), 2000. URL http://www.silicondefense.com/software/spice/.

[15] SANS Institute – Internet Storm Center web site. URL http://isc.sans.org/index.php?on=toptrends.

[16] H. S. Javitz and A. Valdes. The NIDES Statistical Component Description and Justification. Technical Report A010, SRI, 1994.

[17] K. Julisch. Data Mining for Intrusion Detection: A Critical Review. Research Report RZ 3398, IBM Zurich Research Laboratory, 8803 Ruschlikon, Switzerland, February 2002.

[18] T. Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 1995. (Second Extended Edition 1997).

[19] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *SAC '02: Proc. 2002 ACM Symposium on Applied Computing*, pages 201–208. ACM Press, 2002.

[20] K. Labib and V. R. Vemuri. NSOM: A Tool To Detect Denial Of Service Attacks Using Self-Organizing Maps. Technical report, University of California, Davis, 2002.

[21] W. Lee and S. Stolfo. Data mining approaches for intrusion detection. In *Proc. 7th USENIX Security Symposium*, pages 79–94. USENIX Association, 1998.

[22] W. Lee, S. J. Stolfo, and K. W. Mok. A Data Mining Framework for Building Intrusion Detection Models. In *S&P '99: Proc. 20th IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

[23] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):579–595, 2000.

[24] M. V. Mahoney and P. K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *KDD '02: Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 376–385. ACM Press, 2002.

[25] M. V. Mahoney and P. K. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In G. Vigna, C. Kruegel, and E. Jonsson, editors, *RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection*, volume 2820 of *LNCS*, pages 220–237. Springer-Verlag, 2003.

[26] J. McHugh. Testing Intrusion Detection Systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[27] B. V. Nguyen. Self organizing map (SOM) for Anomaly Detection. Technical report, Ohio University, 2002.

[28] M. Ramadas, S. Ostermann, and B. C. Tjaden. Detecting Anomalous Network Traffic with Self-Organizing Maps. In G. Vigna, C. Kruegel, and E. Jonsson, editors, *RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection*, volume 2820 of *LNCS*, pages 36–54. Springer-Verlag, 2003.

[29] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *LISA '99: Proc. 13th USENIX Conference on System Administration*, pages 229–238. USENIX Association, 1999.

[30] Snort Network Intrusion Detection System web site. URL http://www.snort.org.

[31] K. W. G. C. S. J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In A. Valdes and D. Zamboni, editors, *RAID '05: Proc. 8th International Symposium on Recent Advances in Intrusion Detection*, volume 3858 of *LNCS*, pages 227–246. Springer-Verlag, 2006.

[32] K. Wang and S. J. Stolfo. Anomalous Payload-Based Network Intrusion Detection. In E. Jonsson, A. Valdes, and M. Almgren, editors, *RAID '04: Proc. 7th Symposium on Recent Advances in Intrusion Detection*, volume 3224 of *LNCS*, pages 203–222. Springer-Verlag, 2004.

[33] K. Yamanishi, J. Takeuchi, G. J. Williams, and P. Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In *KDD '00: Proc. 6th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 320–324. ACM Press, 2000.

[34] S. Zanero. Analyzing TCP Traffic Patterns using Self Organizing Maps. In F. Roli and S. Vitulano, editors, *ICIAP '05: Proc. 13th International Conference on Image Analysis and Processing*, volume 3617 of *LNCS*, pages 83–90. Springer-Verlag, 2005.

[35] S. Zanero and S. M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *SAC '04: Proc. 19th Annual ACM Symposium on Applied Computing*, pages 412–419. ACM Press, 2004.