Approaches in anomaly-based intrusion detection systems

Damiano Bolzoni, Sandro Etalle

University of Twente,
P.O. Box 2100, 7500 AE Enschede, The Netherlands
{damiano.bolzoni, sandro.etalle}@utwente.nl

Abstract. Anomaly-based network intrusion detection systems can take into consideration packet headers, the payload, or a combination of both. We argue that payload-based approaches are becoming the most effective methods to detect attacks. Nowadays, attacks aim mainly to exploit vulnerabilities at application level: thus, the payload contains the most important information to differentiate normal traffic from anomalous activity. To support our thesis, we present a comparison between different anomaly-based network intrusion detection systems, focusing in particular on the data analyzed by the detection engine to discover possible malicious activities. Furthermore, we present a comparison of two payload-based anomaly-based NIDSes: PAYL and POSEIDON.

1 Introduction

Network intrusion detection systems (NIDSes) are considered an effective second line of defense against network-based attacks directed at computer systems [1, 2], and – due to the increasing severity and likelihood of such attacks – are employed in almost all large-scale IT infrastructures [3].

There exist two main types of intrusion detection systems: signature-based (SBS) and anomaly-based (ABS). SBSes (e.g. Snort [4, 5]) rely on pattern-recognition techniques: they contain a database of *signatures* of known attacks and try to match these signatures against the analyzed data. When a match is found, an alarm is raised. On the other hand, ABSes (e.g. PAYL [6]) first build a *statistical model* describing the normal network traffic, then flag any behaviour that significantly deviates from the model as an attack.

Intuitively speaking, anomaly-based systems have the advantage that (unlike signature-based systems) they can detect zero-day attacks, since novel attacks can be detected as soon as they take place. On the other hand, ABSes (unlike SBSes) require a training phase and a careful setting of the detection threshold (more about this later), which makes their deployment more complex.

Contribution In this paper, we discuss anomaly-based systems, focusing in particular on a specific kind of them: the ABSes *payload-based*. We argue that payload-based systems are particularly suitable to detect advanced attacks, and we describe in detail the most prominent and the most recent of them: respectively Wang and Stolfo's PAYL [6] and our POSEIDON [7].

2 Anomaly-Based Intrusion Detection Systems

In this section, we present the basic working of anomaly-based systems, and we explain the different kinds of ABSes existing. The thesis we try to substantiate here is that, because of the kind of attacks that are carried out nowadays, packed-based and payload-based systems are becoming the most interesting kind of ABS.

Anomaly-based NIDSes can be classified according to:

(a) the underlying algorithm they use,
(b) whether they analyze the features of each packet singularly or of the whole connection, and
(c) the kind of data they analyze. In particular, whether they focus on the packet headers or on the payload.

Regarding the underlying algorithm, Debar et al. [8, 2] define four different possible approaches, but only two of them have successfully employed in the last decade: algorithms based on statistical models and those based on neural networks. The former is the most widely used: according to Debar et al. [8] more than 50% of existing ABSes is statistic-based. In these systems, the algorithm (during the so-called *training phase*) first builds a statistical model of the – legitimate, attack-free – network behaviour; later (in the *detection phase*), the input data is compared to the model using a distance function, and when the distance measured exceeds a given *threshold*, the input is considered *anomalous*, i.e., it is considered an attack. ABSes based on neural networks work in a similar way (they also have a training phase, a detection phase and a threshold), but instead of building a statistical model, they train a neural network which is then in charge of recognizing regular traffic from anomalous one. Good training is of crucial importance for the effectiveness of the system: in particular, the data used in the training phase should be (at least in principle) as *attack-free* as possible: training data must reflect as much as possible the normal system data flow, not including malicious activity. Furthermore, the training phase should be long enough to allow the system to build a faithful model: a too short training phase could lead to a (too) coarse data classification, which – in the detection phase – translates into flagging legitimate traffic too often as anomalous (false positives).

Concerning feature (b) the distinction one has to make is between *packet-oriented* and *connection-oriented* ABSes. A packet-oriented system uses a single packet as minimal information source, while a connection-oriented system considers features of the whole communication before establishing whether it is anomalous or not. Theoretically, a connection-oriented system could use as input the content (payload) of a whole communication (allowing – at least in principle – a more precise analysis), but this would require a long computational time, which would seriously limit the throughput of the system. In practice, connection-oriented systems typically take into account the number of sent/received bytes, the duration of the connection and layer-4 protocol used. According to Wang and Stolfo's benchmarks [6], payload-based ABSes do not show a sensible increase in performance when they also reconstruct the connection, instead of just

considering the packets in isolation. In practice, most ABSes are packet-oriented (see also Table 1).

The last, more practically relevant distinction we can make is between *header-based* and *payload-based* system. Header-based systems consider only packet headers (layer-3 and, if present, layer 4 headers) to detect malicious activities; payload-based systems analyze the payload data carried by the layer-4 protocol; there are also hybrid systems which mix information gathered observing packet headers and (if present) layer-4 payload data. We are going to elaborate on this distinction in the rest of this section. Before we do so, we want to present a table reporting some of the most important ABSes: we select the systems which have been benchmarked with public data sets (either DARPA 1998 [9] or DARPA 1999 [10] data sets, which contain a full dump of the packets, or the KDD 99 [11] data set, which contains only connection meta data).

iSOM [12] uses a one-tier architecture, consisting of a Self-organizing Map [13], to detect two attacks in the 1999 DARPA data set: the first attack against the SMTP service and the other attack against the FTP service. InteligentIDS [14] is based on a Self-organizing Map and extracts information from the connection meta data once it has been reassembled. PHAD [15] combines 34 different values extracted from the packet headers. MADAM ID [16] extracts information from audit traffic and builds classification models (specifically designed for certain types of intrusion) using data mining techniques. The system indicated by SSAD [17] (Service Specific Anomaly Detection) combines different information such as type, length and payload distribution (computing character frequencies and aggregating then them into six groups) of the request. PAYL [6] and POSEIDON [7] detect anomalies only looking at the full payload. Table 1 summarizes the properties of some ABSes.

System	Detection Engine	Semantic Level	Analyzed Data
iSOM	NN	PO + CO	Meta data
IntelligentIDS	NN	CO	Meta data
PHAD	S	PO	H
MADAM ID	S	CO	Meta data
SSAD	S	PO	H + P
PAYL	S	PO	P
POSEIDON	NN + S	PO	P

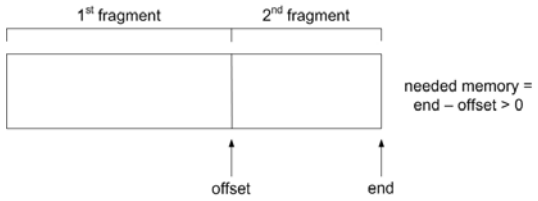
Table 1. Anomaly-based systems: NN stands for Neural Network, S for Statistical model, PO is Packet-Oriented while CO is Connection-Oriented, H and P stand for Headers and Payload respectively

2.1 Payload-based vs header-based approaches

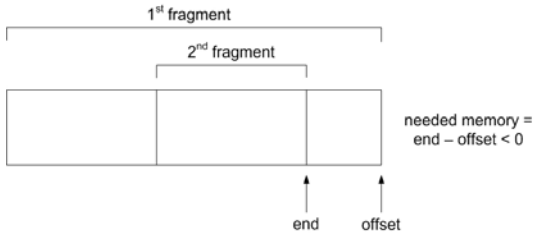
We now elaborate the differences in effectiveness between payload-based and header-based systems. We begin by showing some examples of attacks that can be detected by the systems of one kind, but not by the system of the other kind.

Attacks detectable by header-based systems

Example 1. The teardrop exploit [18] is a remote Denial of Service attack that exploits a flaw in the implementation of older TCP/IP stacks: some implementations of the IP fragmentation re-assembly code on these platforms do not properly handle overlapping IP fragments. Figure 1 shows how the attack takes place: the attacker sends fragmented packets forged so that they overlap each other when the receiving host tries to reassemble them. If the host does not check the boundaries properly, it will try to allocate a memory block with a negative size, causing a kernel panic and crashing the OS.



(a) Correct case



(b) Incorrect case

Fig. 1. A correct and incorrect case of fragment management by the network layer

IDSes can find this attack only by looking for two specially fragmented IP datagrams, analyzing the headers. This attack exploits a vulnerability at the network layer.

Example 2. The Land attack [18] is a remote Denial of Service attack that is effective against some older TCP/IP implementations: the attack involves sending a spoofed TCP SYN packet (connection initiation) with the same source and destination IP address (the target address) and the same (open) TCP port as source and destination.

Some implementations cannot handle this theoretically impossible condition, causing the operating system to go into a loop as it tries to resolve a repeated connections to itself. IDSes detect this attack by looking at packet headers, since TCP SYN segments do not carry any payload. This attack exploits a vulnerability at the transport layer.

Attacks detectable by payload-based systems

Example 3. SQL injection is a technique that exploits vulnerabilities of (web-based) applications which are interfaced to an SQL database: if the application does not sanitize potentially harmful characters first [19], an intruder can *inject* an SQL query in the database, and force the database to output sensitive data (e.g. user passwords and personal details) from database tables, without being authorized. SQL Injections are considered a serious threat and are constantly listed in the "Top Ten Most Critical Web Application Security Vulnerabilities" [20] by "The Open Web Application Security Project".

For instance, the following HTTP request is actually a well-known attack [21] against the Content Management System (CMS) PostNuke [22] that can be used to get hold of the user passwords:

```
http://[target]/[postnuke_dir]/modules.php?op=modload&
name=Messages&file=readpmsg&start=0%20UNION%20SELECT%20
pn_uname,null,pn_uname,pn_pass,pn_p
```

When such an attack is carried out successfully, the output (a database table) is significantly different from the HTML page usually rendered. This attack exploits a vulnerability at the application layer.

Example 4. The PHF attack [23] exploits a badly written CGI script to execute commands with the privilege level of the HTTP server user. Any CGI program which relies on the function *escape_shell_cmd()* may be vulnerable: this vulnerability is manifested by the *phf* program that is distributed with the example code for the Apache web server.

```
http://[target]/cgi-bin/phf?Qalias=x\%0A/bin/cat\%20/etc/passwd
```

The issued request will provide the attacker with the list of system users.

To detect a PHF attack, an intrusion detection system can monitor HTTP requests for invocations of the *phf* command with arguments that specify commands to be run. This attack exploits a vulnerability at the application layer.

Some Conclusions The above examples reflect the unsurprising fact that header-based systems are more suitable to detect attacks directed at vulnerabilities of the network and transport layers; we can also include in this category all the probing techniques used before a real attack takes place (port/host scanning). On the other hand, payload-based systems are more suitable to identify

attacks trying to exploit vulnerabilities at the application level, where sensitive data are stored and most of the systems can be subverted.

Here, we must take notice of the trend that shows that this second kind of attack is increasingly gaining importance: this is due both to the large success of web-based services, and to the fact that attacks at network and transport layers are becoming rare. Because of this, we believe that payload-based system will be increasingly useful in the future. We believe that this trend not only favors payload-based IDS wrt header-based ones, but also anomaly-based systems wrt signature-based ones (we are going to elaborate on this in the conclusions).

We should not forget, however, that payload-based NIDSes cannot function properly in combination with applications or application protocols (e.g. SSH and SSL) which apply data encryption, unless the encryption key is provided. A possible solution to this makes use of a host-based component to access data once they have been decrypted, but this causes an overhead on the monitored host. This problem is going to grow in importance when IPv6 will gradually replace IPv4: in fact, one of the main design issues of IPv6 is the authentication and confidentiality of data (through cryptography).

As we mentioned before, header-based approaches alone do not represent a valid solution to detect modern attacks. In this case, we believe that further research on approaches based on the analysis of connection meta data (connection duration, sent/received bytes, etc.), which are not affected by cryptographic content, should be conducted to verify their application in real environment, since they show to be quite effective in detecting malicious activity with standard data set such as DARPA 1999 and KDD 99.

3 Setting up an ABS

As we have seen, to determine whether a certain input is anomalous or not, an ABS compares it to the model it has: if the distance between some function of the input and the model exceeds a given *threshold*, the input is considered anomalous. This shows that the quality of the model and the value of the threshold have a direct influence on the effectiveness of the ABS. Both the model and the threshold are determined during the system setup (though the threshold could be refined successively). In the rest of this section we elaborate on these two crucial aspects.

Before we do so, we define that the effectiveness of a NIDS is determined by its *completeness* and its *accuracy*.

- $completeness = TP/(TP + FN)$
- $accuracy = TP/(TP + FP)$

Here, TP is the number of true positives, FN is the number of false negatives and FP is the number of false positives raised during a given time frame.

The number of false positives per hour determines the workload of IT personnel: with a hundred thousands input packets per hour (which is a reasonable figure for a web server), a false positive rate of 1% still determines a thousand

false positive per hour, which is more than a typical company can afford to handle. When a system raises too many false positives, then system managers tend to ignore alerts raised. As a matter of fact, a high false positive rate is generally cited as one of the main disadvantages of ABSes.

3.1 Building the Model

The model an ABS refers to should reflect the behaviour of the system *in absence of attacks*, otherwise the ABS may fail to recognize an attack as such. Because of this, the ABS should be trained with a *clean* data set. However, obtaining such a data set is difficult in practice: a casual dump of network traffic is likely to be *noisy*, i.e., to contain attacks.

The standard way to deal with this is by cleaning the data set by manual inspection. This relies completely on the expertise of the IT personnel which must analyze a large amount of data. Clearly, this approach is labour intensive, also because the model of the ABS needs to be updated regularly to adapt to environment changes. The manual inspection can be aided by an automatic inspection using a signature-based IDS, which can pre-process the training data and discover well-known attacks (e.g. web-scanners, old exploits, etc.). A signature-based IDS however will not detect all attacks in the data, leaving the training set with a certain amount of noise.

Nevertheless, we believe that it is possible to clean automatically the data set in such a way that the resulting model is a faithful representation of the legitimate network traffic. Intuitively, we apply an anomaly-based intrusion detection algorithm in which the threshold is set in such a way that we are sure of catching all attacks: this is possible because noise typically forms a small percentage of the total data [24], moreover, its content is typically very different from the content of regular data [25, 26]. The fact that we eliminate also a percentage of the legitimate data with them is – at this stage – not a serious concern.

In our experience clustering techniques from data mining can be quite useful to obtain a good training set. Clustering is the classification of similar objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) shares some common trait, often according to a defined distance measure. Past research applies this concept to detect network attacks [27], but because of the intrinsic limitations of the classification algorithms, it does not achieve a high detection rate. There exists a large number of clustering algorithms which deal with different data types: in our case, we can use clustering algorithms to classify the training data and then disregard the data belonging to the clusters which are not dense enough to be considered significant for the training the model.

3.2 Setting the threshold

The value of the threshold has an obvious and direct impact on the accuracy and completeness: a low threshold yields a high number of alarms, and therefore a low false negative rate, but a high false positive rate. On the other hand, a

high threshold yields a low number of alarms in general (therefore a high number of false negatives, but a low number of false positives). Therefore, setting the threshold requires skill: its "optimal" value depends on the monitored environment and on the distribution of the training data.

In our APHRODITE system [28] we introduce a simple heuristics to set the threshold value when using a *noisy* data set: our experiments show that setting the threshold at $\frac{3t_{max}}{4}$, usually yields reasonably good results; here t_{max} is the maximum distance between the analyzed data and the model observed during the training phase.

4 PAYL and POSEIDON

In this section, we present PAYL and POSEIDON, the first is recognized as the most prominent payload-based anomaly-based NIDS, POSEIDON is the improvement on PAYL we have recently developed.

4.1 PAYL

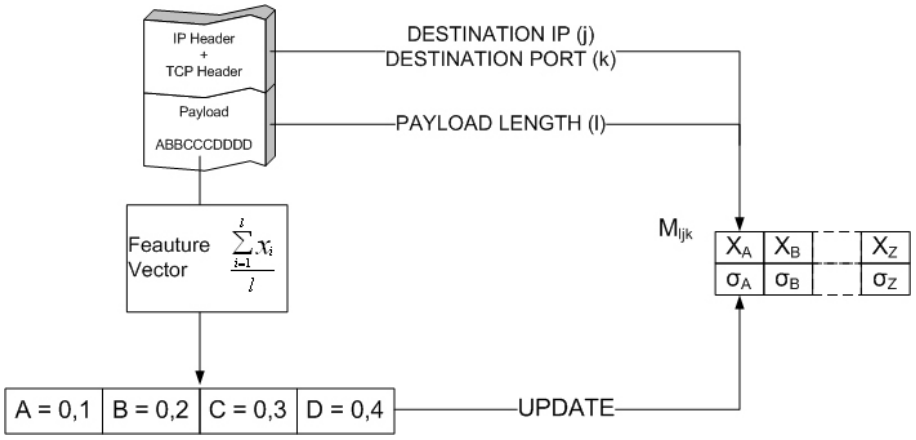


Fig. 2. PAYL architecture

PAYL (Wang and Stolfo [6]) is a system based on a 2-step algorithm. First, packets are classified according to the payload length, then an n-gram [29] analysis is applied to the payload. PAYL works as follows.

During the *training phase*, the training set T is split into a number of disjoint subsets T_{ljk}, where each T_{ljk} contains the packets of length l, destination IP address j and TCP port k. Then PAYL creates statistical models M_{ljk} of each T_{ljk} by first carrying out n-gram analysis [29] of size 1 on each packet of T_{ljk},

and then incrementally storing in M_{ljk} a feature vector containing the average byte frequency distribution together with the variance value of each 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 (payload length can vary between 0 and 1460 in a Local Area Network infrastructure based on Enthernet) 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 collapses similar models. 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 [6] that a reduction in the number of models of up to a factor of 16 can be achieved.

4.2 POSEIDON

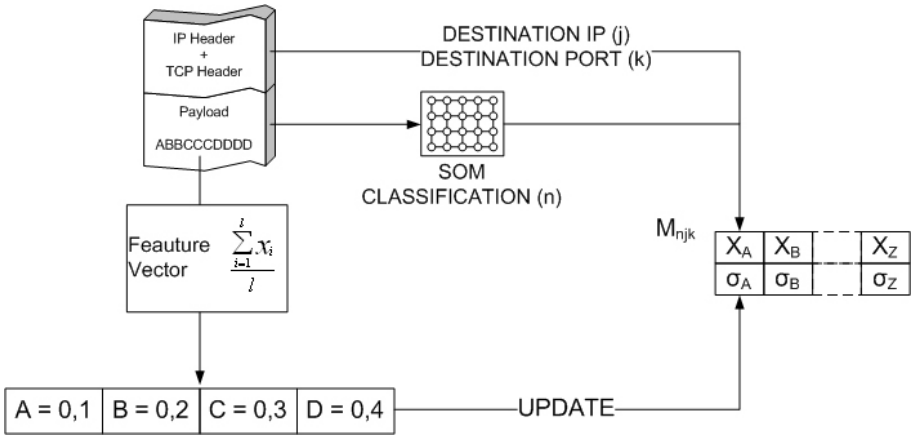


Fig. 3. POSEIDON architecture

The Achille's heel of PAYL lies in the classification phase: a good classification algorithm should produce clusters with high intra-class similarity and high inter-class dissimilarity. Using the payload length information as primary method

to divide packets into clusters, similar payloads could be classified in different clusters because they present a small difference in their length; on the other hand, dissimilar payloads could be classified in the same cluster because they present the same length. This can affect negatively the subsequent n-gram analysis.

To solve this problem, we designed POSEIDON. The design goal was to obtain a good – unsupervised – classification method for network packets (which are high-dimensional data). This is a typical clustering problem which can be tackled using neural networks in general and Self-Organizing Maps (SOM) [13] in particular. SOMs have been widely used in the past both to classify network data and to find anomalies. in POSEIDON, we use them for pre-processing.

The POSEIDON architecture combines a SOM with a modified PAYL algorithm and works as follows. The SOM is used to pre-process each packet, afterwards PAYL uses the classification value given by the SOM instead of the payload length. Instead of using model M_{ijk}, PAYL 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 a 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. Our experiments show that POSEIDON allows to reduce the number of models needed (wrt PAYL) by a factor of 3 while achieving a better accuracy and completeness.

How the SOM works Self-organizing maps are topology-preserving single-layer maps. SOMs are suitable to analyze high-dimensional data and belong to the category of competitive learning networks [13]. 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) are determined 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 compared to all neuron weight arrays w_n with a distance function: 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.

Conclusions Our approach to designing a payload-based ABS 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.

We benchmark our system using the DARPA 1999 data set [10]: this standard data set is used as a reference by a number of researchers (e.g. [15, 12, 6]), and offers the possibility of comparing the performance of various IDSes. Experiments show that our approach achieves a better completeness and accuracy than the original PAYL algorithm: Table 2 reports the results for the four most representative protocols (FTP, Telnet, SMTP and HTTP) inside the data set. Moreover we observe a reduction of models used by PAYL by a factor of 3 when taking advantage of the SOM classification (payload length can vary between 0 and 1460 in a Local Area Network Ethernet-based, while the SOM neural network used in our experiments has less than one hundred nodes).

		PAYL	**POSEIDON**
Number of models used		4065	1622
HTTP	DR	89,00%	100,00%
	FP	0,17%	0,0016%
FTP	DR	95,50%	100,00%
	FP	1,23%	0,93%
Telnet	DR	54,17%	95,12%
	FP	4,71%	6,72%
SMTP	DR	78,57%	100,00%
	FP	3,08%	3,69%
Overall DR with FP $< 1\%$		58,8% (57/97)	73,2% (71/97)

Table 2. Comparison between PAYL and POSEIDON; DR stands for detection rate (completeness), while FP is the false positive rate (accuracy)

5 Conclusions

This paper makes a reasoned case for anomaly payload-based network intrusion detection systems.

Header-Based vs Payload-Based We have argued that header-based approaches are useful in detecting principally attacks at network level, and that most modern remote attacks target vulnerabilities at the application layer: thus, we can no longer rely solely on header-based approaches. Moreover, firewalling systems can easily detect (and discard) well-known malicious packets as well. The fact that header-based approaches manage to achieve high detection rates when benchmarked using the DARPA 1999 data set is – as explained by Mahoney and Chan [30] – often due to the fact that it is possible to tune an IDS which uses some attributes – specifically: remote client address, TTL, TCP options and TCP window size (these present a small range in the DARPA simulation, but have a large and growing range in real traffic) – in such a way that it scores particularly well on this data set. Because most of the attacks to the application significantly differ in content from legitimate traffic (while they are similar when considering header attributes), a payload-based approach is necessary to detect malicious activities.

Signature-based vs Anomaly-Based The fact that that modern attacks are usually directed to weaknesses of the application rather than weaknesses of the underlying system has another important consequence: as we mentioned in the introduction, next to anomaly-based intrusion detection systems, there exist also *signature-based* systems. These system rely on a set of pre-defined signatures (typically defined by the NIDS manufacturer and updated regularly via Internet). Because of the ad-hoc nature of attacks directed at applications, in which it is often possible to modify the syntax of an attack without changing its semantics, signature-based systems are becoming easier to circumvent than anomaly-based systems. For istance, due to the high level of polymorphism presented by SQL Injection attacks, it is impossible to produce few generic signatures to detect them, and most of these attacks will go unnoticed by a SBS.

We believe that the next generation IDS architectures will have to combine signature-based and anomaly-based approaches, as well as network-based with host-based systems: these architectures, supported by adequate correlation techniques, will combine the advantages offered by each approach, improving at the same time the overall completeness and accuracy.

References

1. Bace, R.: Intrusion detection. Macmillan Publishing Co., Inc. (2000)
2. Debar, H., Dacier, M., Wespi, A.: A Revised Taxonomy of Intrusion-Detection Systems. Annales des Télécommunications **55**(7–8) (2000) 361–378
3. Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., Stoner, E.: State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99TR-028, Carnegie-Mellon University - Software Engineering Institute (2000)

4. Roesch, M.: Snort - Lightweight Intrusion Detection for Networks. In: LISA '99: Proc. 13th USENIX Conference on System Administration, USENIX Association (1999) 229–238
5. Sourcefire: Snort Network Intrusion Detection System web site (1999) URL http://www.snort.org.
6. Wang, K., Stolfo, S.J.: Anomalous Payload-Based Network Intrusion Detection. In Jonsson, E., Valdes, A., Almgren, M., eds.: RAID '04: Proc. 7th Symposium on Recent Advances in Intrusion Detection. Volume 3224 of LNCS., Springer-Verlag (2004) 203–222
7. Bolzoni, D., Zambon, E., Etalle, S., Hartel, P.: POSEIDON: a 2-tier Anomaly-based Network Intrusion Detection System. In: IWIA '06: Proc. 4th IEEE International Workshop on Information Assurance, IEEE Computer Society Press (2006) 144–156
8. Debar, H., Dacier, M., Wespi, A.: Towards a Taxonomy of Intrusion-Detection Systems. Computer Networks **31**(8) (1999) 805–822
9. Lippmann, R.P., Cunningham, R.K., Fried, D.J., Garfinkel, S.L., Gorton, A.S., Graf, I., Kendall, K.R., McClung, D.J., Weber, D.J., Webster, S.E., D. Wyschogrod, M.A.Z.: The 1998 DARPA/AFRL off-line intrusion detection evaluation. In: RAID '98: Proc. 1st International Workshop on the Recent Advances in Intrusion Detection. (1998)
10. Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: The 1999 DARPA off-line intrusion detection evaluation. Computer Networks: The International Journal of Computer and Telecommunications Networking **34**(4) (2000) 579–595
11. Bay, S.D., Kibler, D., Pazzani, M., Smyth, P.: The UCI KDD archive of large data sets for data mining research and experimentation. SIGKDD Exploration: Newsletter of SIGKDD and Data Mining **2**(2) (2000) 81–85
12. Nguyen, B.V.: Self organizing map (SOM) for Anomaly Detection. Technical report, Ohio University (2002)
13. Kohonen, T.: Self-Organizing Maps. Volume 30 of Springer Series in Information Sciences. Springer (1995) (Second Extended Edition 1997).
14. Depren, M.O., Topallar, M., Anarim, E., Ciliz, K.: Network Based Anomaly Intrusion Detection using Self Organizing Maps (SOMs). In: SIU '04: Proc. 12th IEEE National Conference on Signal Processing and Applications. (2004) 76–79
15. Mahoney, M.V., Chan, P.K.: 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, ACM Press (2002) 376–385
16. Lee, W., Stolfo, S.J.: A Framework for Constructing Features and Models for Intrusion Detection Systems. ACM Transactions on Information and System Security **3**(4) (2000) 227–261
17. Kruegel, C., Toth, T., Kirda, E.: Service specific anomaly detection for network intrusion detection. In: SAC '02: Proc. 2002 ACM Symposium on Applied Computing, ACM Press (2002) 201–208
18. CERT: Ip Denial of Service Attacks. Technical report, CERT Coordination Center (1997) http://www.cert.org/advisories/CA-1997-28.html.
19. Web Application Security Consortium: Web Security Threat Classification (2005) URL http://www.webappsec.org/projects/threat/.
20. The Open Web Application Security Project: OWASP Top Ten Most Critical Web Application Security Vulnerabilities (2006) URL http://www.owasp.org/documentation/topten.html.

21. Security Reason: PostNuke Input Validation Error (2005) URL http://securitytracker.com/alerts/2005/May/1014066.html.
22. PostNuke: PostNuke Content Managament System (2006) URL http://www.postnuke.com/.
23. CERT: Vulnerability in NCSA/Apache CGI example code. Technical report, CERT Coordination Center (1996) URL http://www.cert.org/advisories/CA-1996-06.html.
24. Portnoy, L., Eskin, E., Stolfo, S.J.: Intrusion detection with unlabeled data using clustering. In: DMSA '01: Proc. of ACM CCS Workshop on Data Mining for Security Applications, 8th ACM Conference on Computer Security (CCS' 01), ACM Press (2002) xx–yy
25. Denning, D.E.: An Intrusion-Detection Model. IEEE Transactions on Software Engineering **SE-13**(2) (1987) 222–232
26. Javitz, H.S., Valdes, A.: The NIDES Statistical Component Description and Justification. Technical Report A010, SRI (1994)
27. Lee, W., Stolfo, S.: Data mining approaches for intrusion detection. In: Proc. 7th USENIX Security Symposium, USENIX Association (1998) 79–94
28. Bolzoni, D., Etalle, S.: APHRODITE: an Anomaly-based Architecture for False Positive Reduction. Technical Report TR-CTIT-06-13, University of Twente, The Netherlands (2006)
29. Damashek, M.: Gauging similarity with n-grams: Language-independent categorization of text. Science **267**(5199) (1995) 843–848
30. Mahoney, M.V., Chan, P.K.: An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In Vigna, G., Kruegel, C., Jonsson, E., eds.: RAID '03: Proc. 6th Symposium on Recent Advances in Intrusion Detection. Volume 2820 of LNCS., Springer-Verlag (2003) 220–237