REAL TIME DETECTION OF NOVEL ATTACKS BY MEANS OF DATA MINING TECHNIQUES *

Marcello Esposito, Claudio Mazzariello, Francesco Oliviero, Simon Pietro Romano, Carlo Sansone
Dipartimento di Informatica e Sistemistica – Università degli Studi di Napoli "Federico II"
Via Claudio 21, 80125 Napoli (Italy)

Keywords: Intrusion Detection, Traffic Features.

Abstract: Rule-based Intrusion Detection Systems (IDS) rely on a set of rules to discover attacks in network traffic. Such rules are usually hand-coded by a security administrator and statically detect one or few attack types: minor modifications of an attack may result in detection failures. For that reason, signature based classification is not the best technique to detect novel or slightly modified attacks. In this paper we approach this problem by extracting a set of features from network traffic and computing rules which are able to classify such traffic. Such techniques are usually employed in off line analysis, as they are very slow and resource-consuming. We want to assess the feasibility of a detection technique which combines the use of a common signature-based intrusion detection system and the deployment of a data mining technique. We will introduce the problem, describe the developed architecture and show some experimental results to demonstrate the usability of such a system.

1 INTRODUCTION

Security is one of the main concerns in the development of new technologies and services over the Internet. The most common and best known tools used to ensure security of companies, campuses and, more in general, of any network, are Firewalls and Antiviruses. Though famous and well known, such tools alone are not enough to protect a system from malicious activities. Basing one's own site's security on the deployment of these instruments relies on the idea that intrusion prevention will suffice in efficently assuring data availability, confidentiality and integrity. Indeed, an iteresting idea about intrusions is that they will sooner or later happen, despite the security policy a network administrator deploys. Based on such assumption, the researchers started to develop instruments able to detect successful intrusions and, in some cases, trace back the path leading to the attack source. This is a more pessimistic, though much more realistic way to look at the problem of network security.

*Research outlined in this paper is partially funded by the Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR) in the framework of the FIRB Project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS)"

2 RELATED WORK

This work has many liaisons with both *intrusion detection* and *data mining*.

As to the first research field, intrusion detection is the art of detecting inappropriate, incorrect or anomalous activity within a system, be it a single host or a whole network. An Intrusion Detection System (IDS) analyzes a data source and, after preprocessing the input, lets a detection engine decide, based on a set of classification criteria, whether the analyzed input instance is normal or anomalous, given a suitable behavior model. Intrusion Detection Systems can be grouped into three main categories: *Network-based Intrusion Detection Systems* (N-IDS) (Vigna and Kemmerer, 1999), *Host-based Intrusion Detection Systems* (H-IDS) (Andersson, 1995) (Tyson, 2000) and *Stack-based Intrusion Detection Systems* (S-IDS) (Laing and Alderson, 2000). This classification depends on the information sources analyzed to detect an intrusive activity. An N-IDS analyzes packets captured directly from the network. By setting network cards in promiscuous mode, an IDS can monitor traffic in order to protect all of the hosts connected to a specified network segment. On the other hand, an H-IDS focuses on a single host's activity: the system protects such a host by directly analyzing the audit trails or system logs produced by the host's operating system. Finally, S-IDS are hybrid systems,

Esposito M., Mazzariello C., Oliviero F., Pietro Romano S. and Sansone C. (2005).
REAL TIME DETECTION OF NOVEL ATTACKS BY MEANS OF DATA MINING TECHNIQUES.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 120-127
DOI: 10.5220/0002536601200127

which operate similarly to a N-IDS, but only analyze packets concerning a single host of the network. They monitor both inbound and outbound traffic, following each packet all the way up the TCP/IP protocol stack, thus allowing the IDS to pull the packet out of the stack even before any application or the operating systems process it. The load each IDS must afford is lower than the total traffic on the network, thus keeping the analysis overhead within reasonable bounds; hypothetically, each host on the network could run a S-IDS.

Intrusion Detection Systems can be roughly classified (Figure 1) as belonging to two main groups as well, depending on the detection technique employed: *anomaly detection* and *misuse detection* (Bace, 2000). Both such techniques rely on the existence of a reliable characterization of what is *normal* and what is not, in a particular networking scenario.

More precisely, anomaly detection techniques base their evaluations on a model of what is normal, and classify as anomalous all the events that fall outside such a model. Indeed, if an anomalous behavior is recognized, this does not necessarily imply that an attack activity has occurred: only few anomalies can be actually classified as attempts to compromise the security of the system. Thus, a relatively serious problem exists with anomaly detection techniques which generate a great amount of false alarms. On the other side, the primary advantage of anomaly detection is its intrinsic capability to discover novel attack types. Numerous approaches exist which determine the variation of an observed behavior from a normal one. A first approach is based on statistical techniques. The detector observes the activity of a subject (e.g. number of open files or TCP state transitions), and creates a profile representing its behavior. Every such profile is a set of "anomaly measures". Statistical techniques can then be used to extract a scalar measure representing the overall anomaly level of the current behavior. The profile measure is thus compared with a threshold value to determine whether the examined behavior is anomalous or not. A second approach, named *predictive pattern generation*, is based on the assumption that an attack is characterized by a specific sequence, i.e. a *pattern*, of events. Hence, if a set of time-based rules describing the temporal evolution of the user's *normal* activity exists, an anomalous behavior is detected in case the observed sequence of events significantly differs from a normal pattern.

Misuse detection, also known as *signature detection*, is performed by classifying as attacks all the events conforming to a model of anomalous behavior. This technique is based on the assumption that an intrusive activity is characterized by a signature, i.e. a well-known pattern. Similarly to anomaly detection, misuse detection can use either statistical techniques or even a neural network approach to predict intrusions. Indeed, the rule-based approach is the most used to detect an attack (SNORT[2](Baker et al., 2004) and Bro[3](Paxson and Terney, 2004)). Intrusions are coded by means of a set of rules: as soon as the examined event matches one of the rules, an attack is detected. A drawback of this approach is that only well-known intrusive activities can be detected, so that the system is vulnerable to novel aggressions; sometimes, few variations in an attack pattern may generate an intrusion that the IDS is not able to detect.

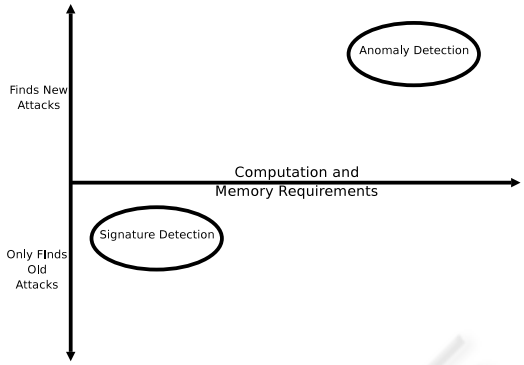

Figure 1: Approaches to Intrusion Detection

The main problem related to both anomaly and misuse detection techniques resides in the encoded models, which define normal or malicious behaviors. Although some recent open source IDS, such as SNORT or Bro, provide mechanisms to write new rules that extend the detection ability of the system, such rules are usually hand-coded by a security administrator, representing a weakness in the definition of new normal or malicious behaviors. Recently, many research groups have focused their attention on the definition of systems able to automatically build a set of models. Data mining techniques are frequently applied to audit data in order to compute specific behavioral models (MADAM ID (Lee and Stolfo, 2000), ADAM (Barbara et al., 2001)).

Coming to the second related research field, we recall that a data mining algorithm is referred to as the process of extracting specific models from a great amount of stored data (Fayyad et al., 1996). Machine learning or pattern recognition processes are usually exploited in order to realize this extraction (SLIPPER[4] (Cohen and Singer, 1999)). These processes may be considered as off-line processes. In fact, all the techniques used to build intrusion detection models need a proper set of audit data. The information must be labelled as either "normal" or "attack" in order to define the suitable behavioral models that

[2] http://www.snort.org
[3] http://www.bro-ids.org
[4] http://www-2.cs.cmu.edu/~wcohen/slipper/

represent these two different categories. Such audit data are quite complicated to obtain. The data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, the 1999 KDD data[5] (Lee and Stolfo, 2000)(Elkan, 2000), is probably the most well-known example of this kind of information, representing a processed version of the DARPA Intrusion Detection Evaluation Program database, collected and managed by the MIT Lincoln Laboratory. The DARPA database contains *tcpdump* data related to seven weeks of network traffic generated over a military emulated LAN. KDD is filled with five million connection records labelled as "normal" or "attack".

3 RATIONALE AND MOTIVATION

Strategies for non punctual intrusion detection often do not take into account the concern of real-time processing of network traffic. Though, an effective IDS should be able to produce the analysis results in time to react and possibly activate countermeasures against malicious behaviors.

The ability to detect an intrusion as soon as it occurs is mandatory for an IDS. The most common types of attacks, e.g. *denial of service*, can be very dangerous if they are not detected in time. Although some IDS store audit data for later analysis, most of them examine such data in real-time so that the system can perform the actions necessary in order to avoid serious problems. Commonly used N-IDS typically analyze packets captured from the network, finding in the current packet the signature of an attack in-progress. However, malicious activity cannot be detected by examining just a single packet: some types of attacks generate in a certain time interval a great amount of packets belonging to different sessions. Hence an efficient detection needs statistical parameters taking into account the temporal relation between sessions. As stated before, Stolfo et al. (Lee and Stolfo, 2000) have defined a set of connection features which summarize the temporal and statistical relations of the connections with reference to each other. These features have been used to create the connection records contained in the KDD database. Several data mining processes use these connection features to extract suitable behavioral models.

Traffic model definition based on an off-line analysis does not consider the unavoidable problems of real-time computation of connection features. The data mining process operates on a database, in which data can be organized in a suitable way in order to compute the features. In real-time intrusion detection, instead, the incoming packets do not contain all of the information needed to compute the connection features, but an appropriate system has to be implemented in order to compute relations among the existing connections. Moreover, off-line analysis does not consider the problem of potential packet losses in the IDS, which has to be taken into account in the case of real time analysis.

Our research aims to develop a framework for real-time intrusion detection. The system we present should be capable to effectively detect intrusions and to operate under a variety of traffic conditions, thus providing an exploitable solution to the issue of real-time analysis. Anomaly detection proves to be the most suitable solution for our purpose, even though such technique has the well known drawback related to the relatively high number of false alarms raised.

Our intrusion detection system can be classified as rule-based. Unfortunately the definition of a rule for every attack is not an efficient solution. On one hand, this approach is not able to detect novel attack patterns; on the other hand, the definition of new attacks has a negative impact both on the computation load and on the average time required to analyze every single packet (hence, the related packet loss problem). In order to overcome the above mentioned drawbacks, by using a set of parameters derived by Stolfo's connection features — which cover a wide range of attack types — it is possible to adopt different data mining processes in order to characterize the attacks by means of different sets of rules.

Summarizing the above considerations, with this work we are interested in the analysis of real-time intrusion detection. To this purpose, we will exploit data mining techniques to design a novel intrusion detection framework. We will present an implementation of the framework and evaluate i ts performance in a real network scenario, by focussing on two main performance figures: packets processing time and system resources needed to compute the connection features.

4 THE REFERENCE MODEL

In this section we present our framework for real-time intrusion detection. The overall model is composed of two parts: the former is the data mining process, which extracts behavioral models from pre-elaborated network traffic, and consists of a database of labelled connection features and a data mining algorithm; the latter is a real-time intrusion detection system which analyzes and classifies network traffic based on the models inferred (Figure 2). In particular, we execute the off-line data mining process on a data set in or-

[5] `http://kdd.ics.uci.edu/`

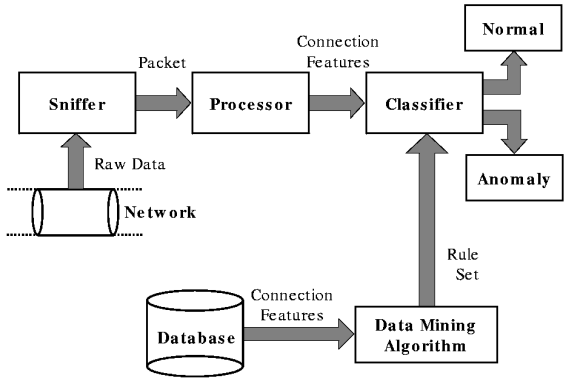

Figure 2: Reference Framework Model

der to extract a set of rules; such a set is then used in a real-time classification process deployed by the IDS that analyzes these pre-computed network data and compares it with informations evaluated by real-time network traffic.

Data mining is part of a more complex KDD (Knowledge Discovery in Databases) process consisting of data analysis and discovery algorithms applied to a database in order to extract high level information — the patterns or the models — able to describe a subset of the data. The models can be applied to unknown data values in order to predict the right class to which they belong. As we emphasized in the previous section, such data mining processes operate on a set of data which has been organized in a suitable fashion (e.g. all the data are identified by a label which explicitly specifies the category they belong to).

In order to implement an efficient classifier, it is important to define a suitable set of features to be extracted from the network traffic contained in the database. The greater the capability of the set of features to discriminate among different categories, the better the classifier. There are three levels at which feature sets may be defined:

- The features may be referred to the single packet captured from the network:

  although this set is easy to compute, it is not able to detect all the potential attack types.

- A set of features related to the entire session which the packet belongs to may be defined:

  this is due to the fact that some intrusions may be realized by means of a sequence of packets belonging to either the same connection or different connections.

- The computed set of features may perform a statistical analysis of the relation between the current session and the other ones:

  this is needed in order to capture intrusions which affect the interrelation among different sessions.

To cope with the aforementioned requirements, we have adopted a model descending from the one proposed by Stolfo. We are interested in TCP, UDP and ICMP traffic. Therefore, a clear definition of the term *connection* is necessary. For a TCP stream the connection can be defined, relying on the protocol specifications, as the collection of messages exchanged between a client process and a server process. For UDP and ICMP we considered each packet as a single, self-contained connection.

The features defined by Stolfo et al. can be classified in tree main groups: *intrinsic* features, *content* features, and *traffic* features. Intrinsic features specify general information on the current session, like the duration in seconds of the connection, the protocol type, the port number (i.e. the service), the number of bytes from the source to the destination, etc. (see Table 1).

Table 1: Intrinsic Features

duration	connection duration (s)
protocol_type	type of transport protocol
service	port number on the server side
src_bytes	bytes from source to destination
dst_bytes	bytes from destination to source
flag	status of the connection
land	land attack
wrong_fragment	number of wrong fragments
urgent	number of urgent packets

The content features are related to the semantic content of connection payload: for example, they specify the number of failed login attempts, or the number of shell prompts (Table 2).

Table 2: Content Features

hot	number of hot indicators
failed_logins	number of failed login attempts
logged_in	successfully logged in
compromised	num compromised conditions
root_shell	root shell is obtained
su	su root command attempted
file_creations	number of file creations
shells	number of shell prompts
access_files	number of file accesses
outbound_cmds	outbound commands in ftp
hot_login	the login belongs to the hot list
guest_login	the login is a guest login

The traffic features can be divided in two groups: the *same host* and the *same service* features. The same host features examine all the connections in the last two seconds to the same destination host as the one involved in the current connection. We also focus on the either the number of such connections, or the rate of connections that have a “SYN” error. Instead, the same service features examine all the connections in the last two seconds to the same destination service as the current one. These two feature sets are de-

fined *time-based* traffic features because they analyze all the events which have occurred in a time interval of two seconds (Table 3); some types of attacks, instead, as the slow probing, may occur every few minutes. Therefore these features might not be able to detect all the attack types. To this aim a new set of traffic features, called *host-based*, has been defined; *same host* and *same service* traffic features are also computed over a window of one hundred connections rather that over a time interval of two seconds. In our framework we will only adopt intrinsic and traffic features. Our purpose is to implement a network-based intrusion detection system, and we deem the content features more suitable for a host-based scenario. Thanks to the access to the operating system's audit trails or system logs, an H-IDS is more efficient in the analysis of the execution of dangerous commands on a single host.

The proposed real-time IDS architecture consists of three components: a *sniffer*, a *processor*, and a *classifier*. The sniffer is the lowest component of the architecture; connected directly to the network infrastucture, this module captures all the packets on the wire. Sniffing is made possible by setting the network card in promiscuous mode. Usually the sniffer also translates raw packets into a human-readable format.

The processor component elaborates the packets captured from the sniffer in order to extract the needed set of features. The main issue of the features computation process is related to the need of keeping up-to-date information about the current connection, as well as the other active sessions. We have to keep in memory a representation of the current network state in order to evaluate the statistical relations among the active connections. Data in memory have to be properly organized in order to reduce the features computation time.

The classifier is the core of the architecture; this component analyzes the current connection features and classifies them. Based on the misuse detection approach, the process of classification uses a set of rules extracted by data mining algorithms. The features are compared against all the rules in the set; when the examined vector of features matches at least one rule, an intrusive action is detected. As to the connection data in the processor component, the rules may be organized in memory in a suitable way in order to reduce the time of analysis.

5 REAL-TIME IDS IMPLEMENTATION ISSUES

The implemented architecture addresses the main requirements of a real-time detection system: monitoring the network traffic in order to extract a set of features from it, as well as behavior classification based on the extracted features. Monitoring, in particular, is the most challenging issue to face from the point of view of a real-time analysis. In our architecture, the monitoring system can be divided into two components: the sniffer that captures traffic from the network, and the processor that computes both the *intrinsic* and the *traffic* features. While in an off-line analysis features computation is simpler, since all the information about connections are stored in a database, in a real time analysis statistic measures have to be be computed every time a new packet is captured from the network (DFP, 2004).

In order to extract features from the traffic, an effective processor must ensure two requirements:

- it holds information about the state of the connection which the analyzed packet belongs to;
- it holds comprehensive information about the traffic flows that have already been seen across the network.

According to the definition proposed in the previous section, every packet can be considered as a single unit that is inserted in a more complex structure, namely the *connection*, and on which the features are computed. While neither UDP nor ICMP traffic requires a heavy load of computation, TCP traffic requires to emulate the TCP state diagram on both the client and the server sides and for every active connection. In particular, when a new packet is captured, the system retrieves information about the connection to which such a packet belongs and updates the connection state of both the client and the server based on the TCP protocol specifications.

In order to compute the statistical relations, information on the past TCP, UDP and ICMP flows is required, including those connections which have been closed. Traffic features, in fact, are computed by analyzing all the connections (either active or expired) having similar characteristics — besides the destination IP address and/or the destination port — as the current one. Every connection has to be kept in memory until it is not needed anymore for other computations.

Our architecture is implemented by means of the open-source N-IDS *Snort*; we have used this system as the base framework on top of which we have built our components. Snort is a lightweight network IDS created by Marty Roesch. Its architecture is made up of four main blocks: a *sniffer*, a *preprocessor engine* that pre-computes of captured packets, a *rule-based detection engine*, and a set of *user output tools*. Thanks to Snort's modular design approach, it is possible to add new functionality to the system by means of *program plugins*. Moreover, Snort provides an efficient preprocessor plugin that reassembles TCP streams and can thus be used to recover the TCP con-

Table 3: Time-Based Traffic Features

Same Host	
count	number of connections to the same host
serror_rate	% of connections with SYN errors
rerror_rate	% of connections with REJ errors
same_srv_rate	% of connections to the same service
diff_srv_rate	% of connections to different services
Same Service	
srv_count	number of connections to the same service
srv_serror_rate	% of connections with SYN errors
srv_rerror_rate	% of connections with REJ errors
srv_diff_host_rate	% of connections to different services

nections status.

We have implemented a new preprocessor plugin which computes the connection features. The main issue we tackled has been the computation of the traffic features, which requires that a proper logical organization of the data is put into place in order to recover information about the past network traffic. Moreover, to assure that the real-time requirement of the system is met, a fast access to stored data is mandatory.

As to the data structures, we have adopted a binary search tree. In the worse case this structure guarantees a performance comparable to that achievable with a linked list from the point of view of search time; performance further improves in case the tree is a static and well-balanced one. Unfortunately, our structure is not a static tree because the connections are not known in advance; though, a self-adjusting binary tree can be adopted in this case in order to balance a dynamic tree.

We have used a Snort library of functions to manage the so-called *Splay Trees*. A Splay Tree is an elegant self-organizing data structure created by Sleator and Tarjan (Sleator and Tarjan, 1985): it actually is an ordered binary tree, in which an item is moved closer to the entry point — i. e. the tree root — whenever it is accessed, by means of a rotation of the item with the parent node. This makes it faster to access the most frequently used elements than the least frequently used ones, without sacrificing the efficiency of operations such as insert and search.

With the above mentioned tree structure, we have implemented two trees, a *Same Host Tree* and a *Same Service Tree* to compute the same host and the same service traffic features, respectively. Every node in the tree is identified by the destination IP address in the first tree, or by the destination service in the second one. In this way, we want to store in the same node information about all the connections that share the same characteristics. In order to compute both the time-based and the host-based traffic features, for every node in the tree we have implemented two linked lists, one for each set. The linked lists contain information like source IP address and/or source port for all the connections that have been identified and that have the same destination IP address and/or the same destination service (Figure 3). The elements of the list, one for every connection, are ordered in time: the first element is the oldest one, the last is the most recent.

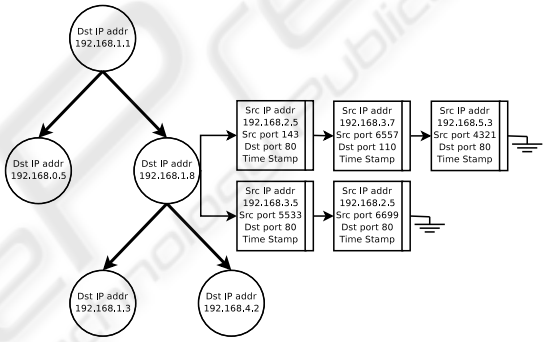

Figure 3: Same-Host Tree Structure

When a new packet is captured from the network, our preprocessor plugin first analyzes the protocol of the packet in order to identify the most appropriate procedure to compute intrinsic features. If the packet belongs to either a UDP or an ICMP traffic, the information required to compute intrinsic features is entirely contained in the packet. In case of TCP traffic, the procedure recovers the session which the packet belongs to in order to determine some crucial information, like the duration of the connection or the number of bytes sent along both directions of the stream, that cannot be directly inferred from the packet. Then, the procedure analyzes the destination IP address and the destination port to compute traffic features. Search operations are performed in both trees: if no preexisting node is found, a new one is created, and the traffic features relative to the current connection are initialized to zero. Otherwise, if a node is already in the tree, the procedure analyzes the two linked lists to compute the statistics for both time-based and host-based traffic features. Every element in the list is analyzed and the statistics are updated. During this process the elements that do not belong neither to a time interval of two seconds, nor

to a window of the latest one hundred connections are pruned off.

6 TESTING THE APPROACH

In this section we evaluate the performance overhead due to the operation of the IDS, pointing out the increase in CPU utilization and memory consumption with respect to the values observed while running Snort without our plugins. Our purpose is to show the affordability of real-time intrusion detection, by means of techniques which are usually employed in off-line analysis. We evaluate both CPU and memory overhead, as well as packet loss ratio. Such tests are deployed in two scenarios: in the former case, we build a testbed to emulate network traffic in a controlled environment; in the latter case, we analyze traffic flowing across the local network at Genova National Research Council (CNR). In this scenario, the most important results concern packet loss analysis. We show that the complexity increase due to the application of our detection techniques does not affect dramatically the percentage of lost packets. Thus we demonstrate the affordability of intrusion detection by means of such techniques. While working on the testbed, we consider the topology depicted in Figure 4.

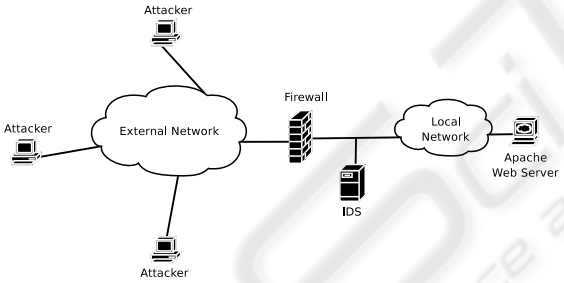

Figure 4: Reference testbed

In order to work in a totally controlled environment, we have to emulate the depicted scenario rather than working in a real network environment; for that purpose, we use another topology which just emulates the one depicted above, as drawn in Figure 5.

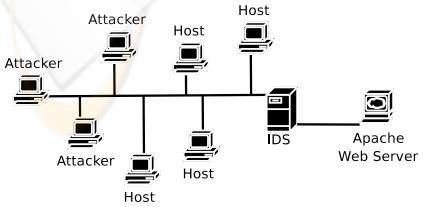

Figure 5: A traffic emulation scenario

Furthermore, we test the IDS using it on a real and heavily loaded network, whose topology is drawn in Figure 6. Such a test is useful to assess the limits of applicability of our plugin, as well as to identify directions for future improvements.

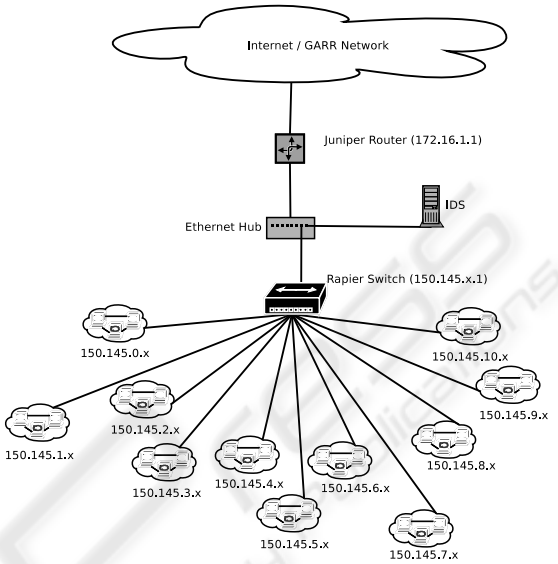

Figure 6: CNR Network Topology

In table Table 4 we see the values of CPU overhead due to the use of Snort alone, versus Snort plus our plugins. The machine operating as IDS in the emulated traffic scenario is equipped with a 1GHz Pentium III CPU and an amount of 256MB RAM, running Mandrake Linux 9.1 as operating system, kernel version 2.4.19. In this case we can point out an almost unperceptible increase in memory consumption (Table 5). The doubling in CPU usage percentage, when using the modified version of Snort with respect to the case of Snort alone, is not such a negative result, since overall CPU usage is still low and under reasonable thresholds, also considering that we are using general purpose, not dedicated, hardware.

Table 4: Average CPU Overhead

	Snort-2.1.0	**Snort + Plugins**
Emulated Traffic	0.12%	0.22%
CNR Traffic	1.16%	2.42%

The extensive test on CNR network also shows a slightly higher CPU usage for the modified version of Snort, still within the limit of 8% overhead. The machine acting as IDS is equipped with a 2GHz Pentium IV, 512MB RAM and RedHat Linux 8.0, using kernel 2.4.18.

Table 5: Memory Overhead

	Snort-2.1.0	**Snort + Plugins**
Emulated Traffic	1.69%	1.70%
CNR Traffic	4.99%	9.46%

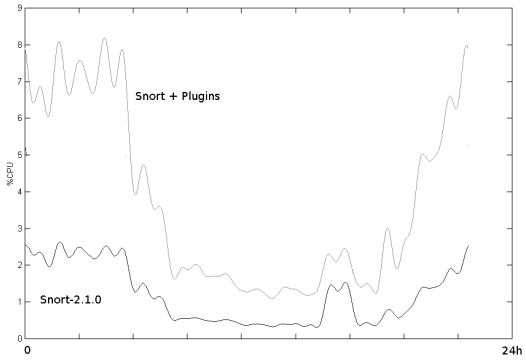

Figure 7: CPU Usage - CNR Network

Once again it is worth pointing out that the results of our measures must be looked at under the perspective of the employment of non dedicated hardware.

Of course, the most interesting indication regards the packet loss ratio. To attain the best results in intrusion detection, the main requirement is not to lose any packets — no matter how much of the system resources we use — if affordable with the available hardware. Such result is sketched in Table 6. In the test deployed using emulated traffic, we notice an increase of less than 10% in packet loss with respect to the plain version of Snort, though the values are lower than the ones obtained by testing the system on a real network. This may be ascribed to the hardware used in the two cases: the setup used in the latter scenario is much more suitable than the one used in the former case. In both cases, anyway, we observe a very low increase in packet loss ratio, showing the feasibility of such a technique.

Table 6: Packet Loss

	Snort-2.1.0	**Snort + Plugins**
Emulated Traffic	0.39%	0.42%
CNR Traffic	0.14%	0.16%

7 CONCLUSIONS AND FUTURE WORKS

This paper shows how it is possible to combine real-time intrusion detection with data mining techniques, while at the same time keeping the system overhead under reasonable thresholds and containing the packet loss ratio within certain boundaries. Future development of this project will involve building rule sets and evaluating their detection capabilities. We may test rulesets computed with different algorithms which make use of various techniques.

The work has also been published on SourceForge[6], to hopefully receive feedback from users and to communicate and cooperate with the Snort community.

ACKNOWLEDGEMENTS

We would like to thank Maurizio Aiello and the staff at CNR laboratory in Genova, Italy, for their cooperation and for providing us with part of the data as well as the equipment used for the tests.

REFERENCES

(2004). *Operation Experience with High-Volume Network Intrusion Detection*. ACM.

Andersson, D. (1995). Detecting usual program behavior using the statistical component of the next-generation intrusion detection expert system (nides). Technical report, Computer Science Laboratory.

Bace, R. G. (2000). *Intrusion Detection*. Macmillan Technical Publishing.

Baker, A. R., Caswell, B., and Poor, M. (2004). *Snort 2.1 Intrusion Detection - Second Edition*. Syngress.

Barbara, D., Couto, J., Jajodia, S., Popyack, L., and Wu, N. (2001). Adam: Detecting intrusion by data mining. pages 11–16. IEEE. Workshop on Information Assurance and Security.

Cohen, W. W. and Singer, Y. (1999). A simple, fast, and effective rule learner.

Elkan, C. (2000). Results of the kdd99 classifier learning. In *SIGKDD Explorations*, volume 1, pages 63–64. ACM.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, pages 37–52.

Laing, B. and Alderson, J. (2000). How to guide - implementing a network based intrusion detection system. Technical report, Internet Security Systems, Sovereign House, 57/59 Vaster Road, Reading.

Lee, W. and Stolfo, S. J. (2000). A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):227–261.

Paxson, V. and Terney, B. (2004). Bro reference manual.

Sleator, D. and Tarjan, R. (1985). Self Adjusting Binary Search Trees. *Journal of the ACM*, 32(3).

Tyson, M. (2000). Derbi: Diagnosys explanation and recovery from computer break-ins. Technical report.

Vigna, G. and Kemmerer, R. (1999). Netstat: a network based intrusion detection system. *Journal of Computer Security*, 7(1).

[6] `http://sourceforge.net/projects/s-predator`