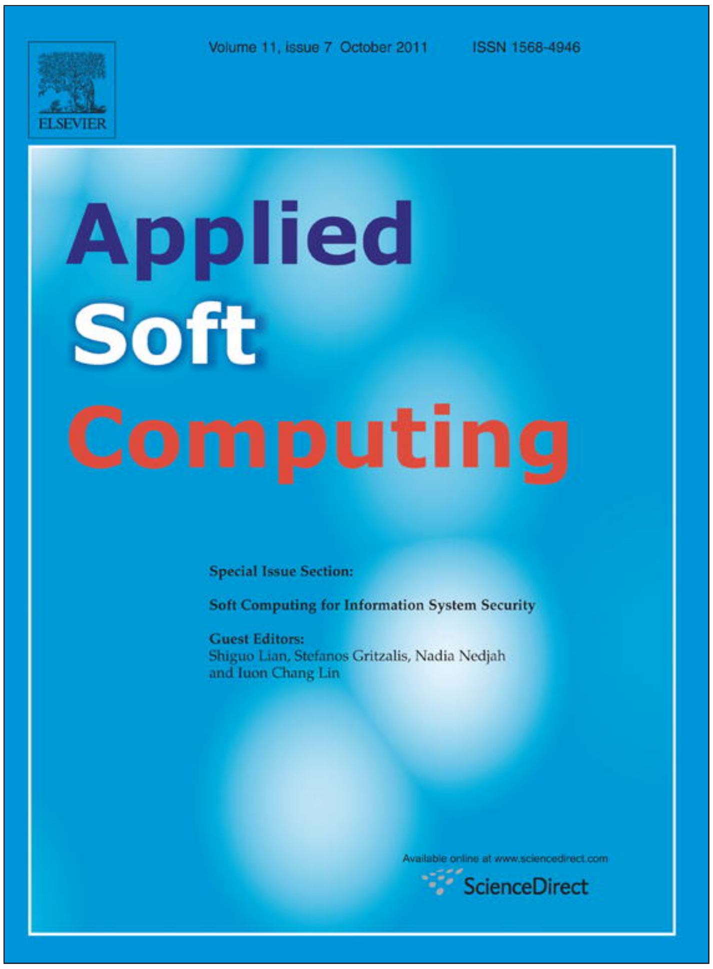

Volume 11, issue 7 October 2011
ISSN 1568-4946
ELSEVIER
Applied
Soft
Computing
Special Issue Section:
Soft Computing for Information System Security
Guest Editors:
Shiguo Lian, Stefanos Gritzalis, Nadia Nedjah
and Iuon Chang Lin
Available online at www.sciencedirect.com
ScienceDirect

Applied Soft Computing 11 (2011) 4384–4398

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Real-time CBR-agent with a mixture of experts in the reuse stage to classify and detect DoS attacks

Cristian I. Pinzón[a], Juan F. De Paz[b], Martí Navarro[c], Javier Bajo[b,*], Vicente Julián[c], Juan. M. Corchado[b]

[a] *Technological University of Panama, Faculty of Computer Systems Engineering, Building No 3, Campus "Dr. Víctor Levi Sasso", Universidad Tecnológica Ave., Panamá City, Panama*
[b] *Departamento Informática y Automática Universidad de Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain*
[c] *Departamento de Sistemas Informaticos y Computacion Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain*

A R T I C L E I N F O

Article history:
Received 28 April 2010
Received in revised form
20 November 2010
Accepted 1 December 2010
Available online 15 December 2010

Keywords:
Service-oriented architectures
Security
Multi-agent systems
Case-based reasoning
Real time

A B S T R A C T

Security is a major concern when service environments are implemented. This has led to the proposal of a variety of specifications and proposals based on soft computing methods to provide the necessary security for these environments. However, most proposed approaches focus only on ensuring confidentiality and integrity, without putting forward mechanisms that ensure the availability of services and resources offered. A considerable number of attack mechanisms can lead to a web service system crash. As a result, the web service cannot allow access to authorized users. This type of attack is a so-called denial of service attack (DoS) which affects the availability of the services and recourses available. This article presents a novel soft computing-based approach to cope with DoS attacks, but unlike existing solutions, our proposal takes into account the different soft computing mechanisms that can lead to a DoS attack. Our approach is based on a real time classifier agent that incorporates a mixture of experts to choose a specific classification technique depending on the feature of the attack and the time available to solve the classification. With this scheme it is possible to divide the problem into subproblems, solving the classification of the web service requests in a more simple and effective way and always within a time bound interval. This research presents a case study to evaluate the effectiveness of the approach and also presents the preliminary results obtained with an initial prototype.

1. Introduction

Confidentiality, integrity and availability are the main objectives of any information security model [1]. With specific regard to availability, the aim is to guarantee that the information, services and available resources are accessible to authorized users [2]. One of the priorities of web services is to guarantee the availability of services and resources. However, all of the specifications that have been proposed for providing security within web services (WS-Security [3], WS-SecurityPolicy [4], WS-Trust [5], WS-SecureConversation [6], etc.) only consider the integrity and confidentiality of messages [7] without much consideration for availability.

One of the increasingly common threats within web service environments and one that jeopardizes the availability factor are denial of service (DoS) [7–10]. This type of attack exploits vulnerable points within the standard components supporting the technology. There are several initiatives within this field: [7,11–18]. However, the main common disadvantage that each of these approaches has is their low capacity to adapt themselves to changes in the patterns. This reduces the effectiveness of these methods when slight variations in the behaviors of known attacks occur or when new attacks appear. Furthermore, most of the existing approaches are based on a centralized perspective. Because of this and the focus on performance aspects, centralized approaches can cause a bottleneck when security is broken, meaning a reduction of the overall performance of the application. Finally, none of these approaches consider the limitations or restrictions in response time.

Response time is a critical aspect in the majority of Internet security systems. With soft-computing systems requiring a response to be given before a specific deadline, as determined by the system needs, it is essential that the execution time for each of the tasks carried out by the soft-computing system is predictable and capable of guaranteeing correct execution within the time needed for the given response. For example, if a service request must be resolved within a specific time, the actual security analysis and execution of the service should not, in the worst case, exceed the determined time. For this reason it is necessary that the response times for analysis, as well as service execution time, are appropriately

* Corresponding author at: Compañía 5, 37002 Salamanca, Spain.
Tel.: +34 639771985; fax: +34 923277101.
E-mail addresses: cristian_ivanp@usal.es (C.I. Pinzón), cofds@usal.es (J.F. De Paz), mnavarro@dsic.upv.es (M. Navarro), jbajope@upsa.es (J. Bajo), vinglada@dsic.upv.es (V. Julián), corchado@usal.es (Juan.M. Corchado).

1568-4946/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2010.12.003

temporal bounded. Furthermore, the agent providing the service and the agent performing the security analysis must both have the necessary mechanisms for executing tasks in a predictable framework, that is, the agent must be prepared for its execution in a real time environment. An agent of this kind must accomplish its goals, responsibilities and tasks with the additional difficulty of temporal restrictions. Thus, a real-time agent can be defined as an agent with temporal restrictions in at least one of its responsibilities. A real-time agent may have its interactions bounded; a modification that will affect all the communication processes in the multi-agent system where the real-time agent is located. Some examples of real time agents are: The ARTIS agent specifically designed to develop Real-Time Systems [19], The ObjectAgent Architecture developed by Princeton Satellites in 2001 [20] and time-aware agents proposed by Prouskas and Pitt in 2002 [21].

This study presents a new agent model with a novel perspective for analyzing and classifying different DoS attack mechanisms. One of the primary characteristics of the proposed agent is its ability to make decisions in real time, making it unique in its conception within the study of DoS attacks. The internal structure of the agent is based on the case-based reasoning (CBR) model, with the main difference being that the different CBR phases are time-bounded, thus enabling its use in real time. CBR can be very suitable for the application in agent reasoning, where similar problems should have similar solutions. However, few of the existing approaches cope with the problem of applying CBR as deliberative engine for agents in MAS with real-time constraints. Some related examples are [22,23] where the CBR phases are faster and more dynamic, allowing for tasks to be carried out quickly. However, in real-time multi-agent systems this concept also implies that deadlines are met on time. Therefore, our CBR approach includes a reasoning cycle that copes with temporal restrictions.

Additionally, the adaptation phase in the CBR system integrated in the agent proposes a new analysis classification model that is carried out by a mixture of experts. This new model makes it possible to divide complicated classification tasks into a series of simple subtasks, so that the fusion of solutions given by the sub-tasks generates the final solution. The concept of a mixture of experts was first proposed by [24]. It involves a system that contains a series of input data that is distributed over a set of expert classifiers. Depending on the time available for performing classification, a set of experts is selected to perform the different analyses. The experts are selected with a multiple method model [25]. Finally the different selected experts generate the predictions and the outputs are fused to generate a new unique result [26].

The agent proposed in this paper aims to deal with DoS attacks in web service environments in a time-bound process. This work is based in our previous research in SQL injection attacks [27,28] in which we developed a SQLMAS multi-agent architecture. The agent model presented is incorporated into the SQLMAS architecture in order to improve the general functioning of the architecture and incorporates real time capabilities and a new classification mechanism based on a mixture of experts.

The rest of the paper is structured as follows: Section 2 presents the problem that has prompted most of this research work. Section 3 shows a general view of the temporal bounded CBR used as deliberative mechanism in the classifier agent. Section 4 is a detailed explanation of the classification model designed. Section 5 describes a set of tests to evaluate our proposal. Finally, the results and conclusion are presented in Section 6.

2. Description of web service security problem

Recently the availability of web services has been threatened by a well known and studied type of attack known as denial of service (DoS) [7,9,10]. This type of attack is generally directed at a particular victim and is fully realized when it manages to deplete the resources within the server (CPU cycles, RAM) following a high number of requests made within a short period of time [29]. This type of attack results in the interruption of server access by authorized users. Furthermore, when the problem of DoS attacks is produced in a web services environment, the risk of one of these attacks being carried out increases considerably. Taking into account the fact that web services are grounded in a series of known standards [30–32], including HTTP, which is the most common means of transporting messages, and XML standard, which is the most commonly used for message coding, we find that the number of vulnerable points increases due to the inherent flexibility of standards and their open nature, which allows for various techniques or attack mechanisms to be carried out.

DoS attack mechanisms at web services level generally take advantage of the costly process that may be associated with certain types of requests. A detailed study with a list of possible web service attacks was presented by [33], and includes attack mechanisms that can affect the availability of web services. Table 1 presents the DoS attack mechanisms analyzed within this study, referencing the previous work of Moradian and Håkansson [33].

In addition to dealing with DoS attacks within web services that evaluate the different possible mechanisms at a granular level, the proposal put forth in this study places great importance on identifying the component within the objective of the web service attack. A DoS attack mechanism can affect the availability of web services to a greater or lesser degree depending on the complexity of the mechanism used and the target component of the attack. One simple example would be to imagine a type of attack in which the parser component is blocked during a period of time. It is probable that the parser component could be activated within minutes of the attack. However if instead of the attack reaching the parser component, it were to reach the database component, it could affect its integrity, clearly resulting in much more serious consequences. The attack would seriously impact the availability of data and actual web service for a much longer period of time. By taking into consideration which component the attack is targeting, it would be possible to extract the relevant information, allowing us to identify hidden vulnerabilities within the web service components and increase ability to make the decisions required for dealing with a real attack.

Finally, it is important to understand that the focus of our proposal centers on the classification of web service requests through SOAP messages. This classification will be carried out by a classifier agent that performs classification based on the data extracted from the structure and content of the messages. The classifier agent could be in charge of analyzing the DoS attack mechanisms (Replay attacks, XDoS attacks), basing the detection of an attack on the analysis of the behavior of traffic directed at the server where the web services are located. However, in our proposal that task would be delegated to a second agent, whose job is to free the classifier agent of that responsibility. This second agent would work alongside the classifier agent to detect different DoS attack mechanisms, forming part of an architecture that is being developed. The following section reviews the most relevant points associated with the subject.

Bebawy et al. propose the “Nedgty” tool [11], which is based on a web service firewall model. The targeted optimal operating system for Nedgty is the Linux OS. Nedgty secures web services by applying business specific rules in a centralized manner. It works at application level as a stand-alone application and its design is a hybrid of a fully fledged proxy. This solution secures web services by intercepting packets going to the server, determining the specific web service packets, and checking them for any malicious content. In addition, it filters out unauthorized requests that originate from

Table 1
Types and targets of attacks including level of damage.

	Attack mechanism	Description	Objective component	Damage level
Evaluating content and structure of SOAP messages	Recursive Payloads	A message written in XML can harbor as many elements as required, complicating the structure to the point of overloading the parser requiring a high amount of memory and processing resources	Parser	Low
	Oversize payloads	When executed, it reduces or eliminates the availability of a web service while the CPU, memory or bandwidth are being tied up by a massive mailing with a large payload	Parser	Low
	Schema poisoning	An attacker compromises XML Schema and replaces it with similar, but modified one	Parser	Low
	Buffer overflow	This attack targets the SOAP engine through the web server. An attacker sends more input than the program can handle, which can cause the service to crash	Web service application	Low
	XML injection	Any element that is maliciously added to the XML structure of the message can reach and even block the actual Web service application	Web service application	Low
	SQL injection	An attacker inserts and executes malicious SQL statements into XML	Database	High
	Xpath injection	An attacker forms SQL-like queries on an XML document using Xpath to extract an XML database	XML database	High
Evaluating traffic	Replay attacks	To overload web service an attacker steals messages and sends them repeatedly	Parser/web service application	Low
	XML denial of service attack	An attacker trying to prevent legitimate users from accessing a service by flooding the service with thousands of requests		Low

unauthorized IP addresses. A XML Firewall is proposed by [13]. The architecture of the XML Firewall is divided into three modules: Core engine, Administrative Interface, and database. The Core engine is the main component that processes and handles SOAP messages. Messages that are sent to a web service are intercepted and parsed to check the validity and authenticity of content. If the content of the messages does not conform to the policies that have been set, the messages will be dropped by the firewall. Three successfully implemented filtering policies, message size filtering, syntax parsing, and XML Schema validation, have been tested with valid and invalid SOAP messages. Gruschka et al. [7] propose an application level gateway system "Checkway". They focus on a full grammatical validation of messages by Checkway before forwarding them to the server. To do this, they consider that web service messages are XML documents, which are usually defined by an XML Schema and written in XML Schema definition language, a grammar language for XML. Checkway generates an XML Schema from a web service description and validates all web service messages against this schema. The approach presents a centralized model oriented to detect specific types of attacks inside web services. An adaptive framework for the prevention and detection of intrusions was presented in [15]. Based on a hybrid approach that combines agents, data mining and diffused logic, it is designed to filter attacks that are either new or already known. Agents that act as sensors are used to detect violations to the normal profile using data mining techniques such as clustering, association rules and sequential association rules. The anomalies are then further analyzed using fuzzy logic to determine genuine attacks so as to reduce false alarms. If an attack is being detected, a specific component will act to prevent the attack from happening. Sidharth and Liu [34] propose an Integrated Application and Protocol-based Framework to tackle existing WS-Security problems. The proposed IAPF techniques are targeted to be part of the design and implementation structure of a web service. In the IAPF approach, the first step involves providing protection against vulnerabilities in the UDDI protocol. The second step involves protecting vulnerabilities in the WSDL protocol. In the third step, comprehensive protections are built using techniques such as WS-Security to protect vulnerabilities in the SOAP protocol for end to end communication between two entities. In the fourth step, protection mechanisms are built to protect SOAP web services that need to be exposed openly to third party consumers. The IAPF approach has been presented as a theoretical proposal. An approach to handling DoS attacks using a twofold mechanism is presented by [16]. First, an admission control is performed to limit the number of concurrent clients served by the online service. Admission control is based on hiding ports, which renders online service invisible to unauthorized clients by hiding the port number on which the service accepts incoming requests. Second, a congestion control is performed on admitted clients to allocate more resources to good clients. Congestion control is achieved by adaptively setting a client's priority level in response to the client's requests in a way that can incorporate application-level semantics. Experiments show that the techniques incur low performance overhead. In addition, the proposed techniques can be easily deployed in existing web/application servers. An approach to countering DDoS and XDoS attacks against web services is presented by [17]. The system requests message authentication and validation before the requests are processed by the web service providers. The scheme has two modes: normal mode and the under-attack mode. A component called "operations provider" decides which mode the system works in. In the under-attack mode, the service requests need to be authenticated and validated before being processed. Since the system is constructed from web services, it can be formed and reconfigured easily. Finally, a recent solution proposed by [18] presents a Service Oriented Traceback Architecture (SOTA) to cooperate with a filter defense system, called XDetector. XDetector is a Back Propagation Neural Network trained to detect and filter XDoS attack messages. SOTA is a traceback system based on web services and able to trace back to the source of the malicious message. Once an attack has been discovered and the attacker's identity known, XDetector can filter out the attack messages.

The approach provided in our study exceeds those of previous soft computing-based studies with regards to the following characteristics:

- Time response: our focus makes it possible to perform a real time analysis, which guarantees a response within the time restrictions that are associated with the service request. None of the previously mentioned studies give much attention to this factor, which may affect the quality of the web service.
- Adaptive ability: our approach includes different types of intelligent agents designed to learn and adapt to changes in attack patterns as well as new attacks. This includes a temporally bound CBR engine and a mechanism known as a mixture of experts that

can assign the most appropriate techniques for identifying the type of attack.
- Scalability: our approach is capable of growing (by means of the instantiation of new agents) according to the needs of its environment.

In addition to these capabilities, once our classifier agent is integrated into a multi-agent architecture, as expected, we will obtain new advantages such as:

- Distributed approach: our classifier agent will be integrated into a multi-agent architecture that can execute tasks derived from the classification process in a distributed way.
- Balancing the workload: the use of a multi-agent architecture with a distributed hierarchy makes it possible to distribute the classification task load throughout the various layers of the hierarchical architecture.
- Tolerance for failure: the hierarchical design used in the multi-agent architecture can facilitate error recovery through the instantiation of new agents.
- Ubiquity: once the classifier agent has been integrated within the multi-agent architecture, it will be capable of providing an ubiquitous alert mechanism to notify security personnel in the event of an attack.

Our proposal provides a much more efficient classification once the system acquires experience, and a reasonably low time bounded response. The architecture proposed presents novel characteristics that have not been considered in previous approaches. The next section presents the real time agent model used to implement the classifier agent.

3. Real time agent and case-based reasoning (CBR)

This section presents the new model of real time agents with advanced reasoning capabilities. The agent combines learning and adaptation capabilities in order to provide a case-based reasoning system capable of being executed under time bounded restrictions that occur in real time scenarios.

A real time agent is one that is able to support tasks that should be performed within a restricted period of time [35]. This characteristic justifies its use in real time systems. In this type of environment, the validity of the solution is determined not only by its correct execution, but by its ability to be carried out within the allotted time frame [36].

The main problem in the architecture of a real time agent (RTA) is with the deliberation process. This process may use Artificial Intelligence (AI) techniques as problem-solving methods to compute more intelligent actions. If this is the case, it is difficult to know the time required, because it can either be unbounded or have a high variability. If the agent has to operate in a real-time environment, the agent complexity required to achieve any or all of these features is greatly increased. Thus a RTA requires an efficient integration of high-level, deliberative processes within reactive processes. When using AI methods, it is necessary to provide techniques that allow their response times to be bounded. These techniques are mainly based on well-known Real-Time Artificial Intelligence System (RTAIS) techniques [37,38].

Therefore, it would be interesting to integrate complex deliberative processes for decision-making in real-time agents and to do so in a simple and efficient way. Some of the most important features of agents are their ability to work autonomously, to adapt to the environment, to reason, to learn, to predict the future effect of the performed actions, and to predict the future behavior of the environment. Intelligent agents may use a lot of reasoning mechanisms to achieve these capabilities, including planning techniques [39] or case-based reasoning (CBR) techniques [40].

There are many CBR applications to control at least some aspects of the deliberative process of agents in a MAS developed for specific purposes [40]. The main assumption in CBR is that similar problems have similar solutions. Therefore, when a CBR system has to solve a new problem, it retrieves precedents from its case-base and adapts their solutions to fit the current situation. This reasoning methodology greatly resembles the way people reason about their experiences. CBR can thus be very suitably applied in agent reasoning, where similar problems should have similar solutions. However, few of the existing approaches cope with the problem of applying CBR as a deliberative engine for agents in MAS with real-time constraints. If we want to use CBR techniques as a reasoning mechanism in real-time agents, it is necessary to adapt these techniques to be executed so that they guarantee real-time constraints. In real-time environments, CBR phases must be temporally bounded to ensure that solutions are produced on time, giving the system a case-based behavior that is time bound and deliberative.

As a first step, we propose a modification of the classic CBR cycle, adapting it so that it can be applied in real-time domains. First, we group the four reasoning phases that implement the cognitive task of the real-time agent into two stages defined as: the learning stage, which consists of the revise and retain phases; and the deliberative stage, which includes the retrieve and reuse phases. Each phase will schedule its own execution time. Therefore, the designer can choose to either assign more time to the deliberative stage, or keep more time for the learning stage (and thus for the design agents that are more sensitive to updates). These new CBR stages must be designed as an anytime algorithm [41], where the process is iterative and each iteration is time-bounded and may improve the final response.

In accordance with this, our time bounded CBR cycle (TB-CBR) will operate in the following manner. To begin, the main difference that can be observed between the classic CBR cycle and the TB-CBR cycle is the starting phase. Recent changes in the case-base will commonly affect the potential solution that the CBR cycle is able to provide for a current problem. Therefore, the TB-CBR cycle starts at the learning stage, checking if there are previous cases waiting to be revised and possibly stored in the case-base. In our model, the solutions provided at the end of the deliberative stage will be stored in a solution list while a feedback about their utility is received. When each new CBR cycle begins, this list is accessed and while there is enough time, the learning stage of those cases whose solution feedback has been recently received is executed. If the list is empty, this process is omitted.

After this, the deliberative stage is executed. The retrieval algorithm is used to search the case-base and retrieve a case that is similar to the current case (i.e., the one that characterizes the problem to be solved). Each time a similar case is found, it is sent to the reuse phase where it is transformed into a suitable solution for the current problem using a reuse algorithm. Therefore, at the end of each iteration of the deliberative stage, the TB-CBR method is able to provide a solution for the problem at hand and this solution can be improved in subsequent iterations if the deliberative stage has enough time to perform them. Hence, the temporal cost of executing the cognitive task is greater than or equal to the sum of the execution times of the learning and deliberative stages (1):

$$\begin{aligned}&T_{\text{cognitiveTask}} \geq t_{\text{learning}} + t_{\text{deliberative}}\\ &t_{\text{learning}} \geq (t_{\text{revise}} + t_{\text{retain}}) \times n\\ &t_{\text{deliberative}} \geq (t_{\text{retrieve}} + t_{\text{revise}}) \times m\end{aligned} \tag{1}$$

where t_{learning} and $t_{\text{deliberative}}$ are the total execution time of the learning and deliberative stages; t_x is the execution time of the phase x and n and m are the number of iterations of the learning and deliberative stages, respectively.

```
     Input: $t_{max}$
1.1  ($t_{learning}$,$t_{deliberative}$) $\longleftarrow$ timeManager($t_{max}$)
1.2  if solutionQueue $\neq \emptyset$ then
1.3      while enoughTime($t_{now}$,$t_{revise}$,$t_{retain}$,$t_{learning}$) and
         solutionQueue $\neq \emptyset$ do
1.4          $r \longleftarrow$ pop(solutionQueue)
1.5          $\{$adequate $\longleftarrow$ analysesResult($r$)$\}^{\leq t_{revise}}$
1.6          if adequate then
1.7              $\{$retainResult($r$)$\}^{\leq t_{retain}}$
1.8          end
1.9      end
1.10 end
1.11 if problemQueue $\neq \emptyset$ then
1.12     problem $\longleftarrow$ pop(problemQueue)
1.13     repeat
1.14         $\{$cases $\longleftarrow$ push(search(adaptProblem(problem)))$\}^{\leq t_{retrieve}}$
1.15         $\{$solution $\longleftarrow$ adaptSolution(cases,problem)
1.16         bestSolution $\longleftarrow$ bestSolution(solution,bestSolution) $\}^{\leq t_{reuse}}$
1.17     until $\neg$enoughTime($t_{now}$,$t_{retrieve}$,$t_{reuse}$,$t_{deliberative}$) ;
1.18     solutionQueue $\longleftarrow$ push(bestSolution)
1.19     return bestSolution
1.20 end
```

Algorithm 1. Time bounded CBR algorithm.

According to this temporal restriction, a first view of the TB-CBR algorithm can be seen in Algorithm 1. This algorithm can be launched when the real-time agent considers it appropriate and there is enough time for it to be executed. The real-time agent indicates to the TB-CBR the maximum time ($t_{\max}$, where $t_{\max} \geq t_{\text{cognitiveTask}}$) that is available to complete its execution cycle. The time $t_{\max}$ must be divided between the learning and the deliberative stages to guarantee the execution of each stage. The *timeManager* ($t_{\max}$) function is in charge of completing this task. Using this function the designer must specify how the real-time agent acts in the environment. The designer can assign more time to the learning stage if it desires a real-time agent with a greater capacity to learn. Otherwise, the function can allocate more time to the deliberation stage. Regardless of the type of agent, the *timeManager* function should allow sufficient time for the deliberative stage to ensure a minimal answer.

The first phase of the algorithm executes the learning stage. This stage is executed only if the real-time agent has the solutions of previous executions stored in the *solutionQueue*. The solutions are stored just after the end of the deliberative stage. The deliberative stage is only launched if the real-time agent has a problem to solve in the problemQueue. This configuration allows the agent to launch the TB-CBR system solely in order to learn (no solution is needed and the agent has enough time to reason about previous decisions), to deliberate (there are no previous solutions to consider and there is a new problem to solve) or both.

The following section presents an agent that incorporates the TB-CBR mechanism to perform security analysis for web service requests.

4. Improved security services by means of a real-time agent

This section presents an agent specially designed to incorporate an adaptation of the previously mentioned TB-CBR model as a reasoning engine. The aim of the reasoning model is to obtain a soft computing system to facilitate real time decision making in a robust manner and low solution cost. The learning phase is eliminated since it is now performed by human experts. The TB-CBR agent utilizes a global case base, which avoids any duplication of information compiled from the cases or any information contained in the results of the analysis. Tables 2–4 show the structure of the cases. Table 2 shows the fields recovered from the analysis of service request headers. Table 3 shows the fields associated with the analysis of service requests that were obtained after the analysis performed by the parser application.

Finally, Table 4 shows the information obtained after analyzing the service requests.

From the information contained in these tables, it is possible to obtain the global structure of the cases. In this way, the information of each case can be represented by the following tuple:

$$c = \left(\left\{ \frac{h_i}{i} = 1 \ldots 6 \right\} \cup \left\{ \frac{p_j}{j} = 1 \ldots 27 \right\} \cup \left\{ \frac{x_{lm}}{l} = 1 \ldots 6, \quad m = 1 \ldots 2 \right\} \right) \tag{2}$$

When the system receives a new request, the TB-CBR agent performs an analysis that can determine whether it is an attack, in which case it identifies the type of attack. The following sections describe the different stages of the deliberative stage for the TB-CBR model and identify the analysis and classification functions of possible attacks on the system. These stages have been temporally bounded in order to be used in situations containing temporal restrictions.

Table 2
Header definitions.

Fields	Type	Variable
IDService	Int	h_1
Subnet mask	String	h_2
SizeMessage	Int	h_3
NTimeRouting	Int	h_4
LengthSOAPAction	Int	h_5
TFMessageSent	Int	h_6

Table 3
Definition of fields recovered by the parser.

Description	Fields	Variable
Number of header elements	NumberHeaderElement (Int)	p_1
Number of elements in the body	NElementsBody (Int)	p_2
Greatest value associated to the nesting elements	NestingDepthElements(Int)	p_3
Greatest value associated to the repeated tag within the body	NXMLTagRepeated (Int)	p_4
Greatest value associated with the leaf nodes among the declared parents	NLeafNodesBody (Int)	p_5
Greatest value of the associated attributes among the declared elements	NAttributesDeclared (Int)	p_6
Type of SQL command	Command_Type (Int)	p_7
Number of times that the *AND* operator appears in the string	Number_And (Int)	p_8
Number of times that the *OR* operator appears in the string	Number_Or (Int)	p_9
Number of times that the *Group By* function appears	Number_GroupBy (Int)	p_{10}
Number of times that the *Order By* function appears	Number_OrderBy (Int)	p_{11}
Number of times that the *Having* function appears	Number_Having (Int)	p_{12}
Number of *Literals* declared in the string	Number_Literals (Int)	p_{13}
Number of times that the *Literal Operator Literal* expression appears	Number_LOL(Int)	p_{14}
Length of the SQL string	Length_SQL_String (Int)	p_{15}
Greatest value associated with the length of the string among the elements or attributes within the body	LengthStringValueBody (Int)	p_{16}
Total number of incidences during the parsing process	TotalNumberIncidenceParsing(Int)	p_{17}
Reference to an external entity	URIExternalReference (Boolean)	p_{18}
Number of variables declared in hte Xpath expression	XpathVariablesDeclared (Int)	p_{19}
Number of elements affected in the consulted node	XpathNumberElementAffected(Int)	p_{20}
Number of literals declared in the XQuery Statement	XpathNumberLiteralsDeclared (Int)	p_{21}
Number of times the And operator appears in the XQuery Statement	XpathNumberAndOperator (Int)	p_{22}
Number of times the Or operator appears in the XQuery Statement	XpathNumberOrOperator (Int)	p_{23}
Number of functions declared in the XQuery Statement	XpathNumberFunctionDeclared (Int)	p_{24}
Lentgh of the XQuery Statement in a SOAP message	XpathLenghtStatement (Int)	p_{25}
Cost of processing time (CPU)	CPUTimeParsing (Int)	p_{26}
Cost of memory size (KB)	SizeKbMemoryParser (Int)	p_{27}

Table 4
Fields stored as results.

Fields	Type	Variable
Probability oversize payload	Real	x_{11}
Attack oversize payload	Boolean	x_{12}
Probability recursive parsing	Real	x_{21}
Attack recursive parsing	Boolean	x_{22}
Probability buffer overflow attack	Real	x_{31}
Attack buffer overflow attack	Boolean	x_{32}
Probability XML injection attack	Real	x_{41}
Attack buffer overflow attack	Boolean	x_{42}
Probability Xpath injection attack	Real	x_{51}
Attack Xpath injection attack	Boolean	x_{52}
Probability SQL injection attack	Real	x_{61}
Attack SQL injection attack	Boolean	x_{62}

4.1. Retrieve

In the retrieve phase, the real time agent recovers the cases that it will use to perform classification. The time needed to recover the different cases to be used is clearly defined and temporally bounded. The retrieval time for the cases depends on the number of cases in the case base. If the number is known, it is easy to predict how much execution time will be used to recover the cases. The asyntotic cost is linear ($O(n)$).

The cases that have been retrieved during this phase are selected according to the information obtained from the headers of the packages of the HTTP/TCP-IP transport protocol from the new case. The information retrieved corresponds to the service description fields, and the service requestor's subnet mask. Assuming that the newly introduced case is represented by c_{n+1}, the case c_{n+1} is defined by the following tuple: $c_{n+1} = (\{h_i / i = 1 \ldots 6\})$. The new case does not initially contain information related to the parser.

$$c_{\cdot h_1 h_2} = f_s(C) = \left\{ c_{j\cdot h_1 h_2} \in \frac{C}{c_{j\cdot h_1}} = c_{n+1\cdot h_1} \cap c_{j\cdot h_2} = c_{n+1\cdot h_2} \right\} \tag{3}$$

where $c_{j\cdot h_1}$ represents the case j and h_1, a property that is determined according to the data shown in Table 2, C represents the set of cases, and f_s the retrieval function.

Table 5
Combination of techniques and the execution time associated with each.

	Oversize payload		Recursive parsing		Buffer overflow attack		XML injection attack		WCET
	Light	Heavy	Light	Heavy	Light	Heavy	Light	Heavy	ms
C_1	✓		✓		✓		✓		6.12E−01
C_2		✓	✓		✓		✓		8.86E−01
C_3	✓			✓	✓		✓		8.89E−01
C_4		✓		✓	✓		✓		8.89E−01
C_5	✓		✓			✓	✓		8.94E−01
C_6		✓	✓			✓	✓		1.16E+00
C_7	✓			✓		✓	✓		1.16E+00
C_8		✓		✓		✓	✓		1.17E+00
C_9	✓		✓		✓			✓	1.17E+00
C_{10}		✓	✓		✓			✓	1.17E+00
C_{11}	✓			✓	✓			✓	1.17E+00
C_{12}		✓		✓	✓			✓	1.44E+00
C_{13}	✓		✓			✓		✓	1.44E+00
C_{14}		✓	✓			✓		✓	1.44E+00
C_{15}	✓			✓		✓		✓	1.45E+00
C_{16}		✓		✓		✓		✓	1.72E+00

Table 6
Techniques associated with attacks.

	Light technique	Heavy technique
Oversize payload	Decision tree	Neural network
Recursive parsing	Naïve Bayes	Neural network
Buffer overflow attack	Decision tree	Neural network
XML injection attack	SMO	Neural network
Xpath injection attack		Neural network
SQL injection attack		Neural network

Table 7
Inputs associated with different attack mechanisms.

Fields	Variable
Recursive parsing	$\{h_3, h_4, h_5, h_6, p_c\}$
Oversize payload	$\{h_3, h_4, h_5, h_6, p_c\}$
XML injection attack	$\{h_3, h_4, h_5, h_6, p_c\}$
Buffer overflow attack	$\{h_3, h_4, h_5, h_6, p_c\}$
SQL injection attack	$\{p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}\}$ U $\{h_3, h_4, h_5, h_6, p_c\}$
Xpath injection attack	$\{p_{19}, p_{20}, p_{21}, p_{22}, p_{23}, p_{24}\}$ U $\{h_3, h_4, h_5, h_6, p_c\}$

In the event that the retrieved set is empty, the process continues with the retrieval of the messages only without considering the subnet mask.

The process of parsing is carried out at the beginning of the retrieval phase so that the information is available and can minimize the waiting time during the reuse phase. If the parser exceeds the time limit set for analyzing the request, it assumes that the request is malicious and rejects it. By keeping this restriction in mind it is possible to work in real time and also guarantee the integrity of the parser when facing malicious requests.

4.2. *Reuse*

At the beginning of the reuse phase, a number of different techniques of soft computing are applied to the set of retrieved cases, making it possible to determine if the service request is a DoS attack, and if so, what kind of attack it is.

Each type of attack in our proposal (except for Xpath and SQL injection attacks) can be analyzed by two different techniques. The first is known as the light technique and is usually a detection algorithm with a low temporal cost, but of low quality as well. Using the heavy technique, the result of the analysis is much more exact, but it requires a much higher amount of execution time. Using these techniques to analyze an attack allows the real time agent to apply the one that is best suited to its needs, without violating the temporal restrictions that should be considered when executing the deliberative stages. The objective is to exploit tolerance for imprecision, uncertainty to achieve tractability, robustness and low solution cost. Obviously, the more time that the real time agent has available to execute the reuse phase, the more detailed the analysis of the request can be. However, in many cases the real time agent has a limited amount of time to complete the analysis and must select which combination of techniques will allow it to complete a full analysis within a period of time that does not exceed the time limit. Planning which tasks will be used is based on a generalization of the estimated process, known as design-to-time [25] planning, which assumes that multiple methods exist to complete various tasks and the problem consists of designing a solution that uses all available resources to maximize the quality of the response within the available time. In order to determine which set of techniques provides the best solution, it is necessary for the length and quality associated with each technique to be predictable, as seen from a global perspective that includes all possible combinations of techniques. Table 5 shows the different combinations of techniques that can be selected and the worst case execution time. Xpath and SQL injection attack mechanisms have only one attack technique that provides the best solution. This is because the seriousness of the harm that each of these attack mechanisms can inflict makes it necessary to perform a more thorough analysis.

Algorithm 2 is used to select the combination of techniques that produce the most optimal results given the time restraint. This algorithm uses the adaptProblem function shown in Algorithm 1 to select the optimal combination given the amount of time available. Furthermore, a prior selection of available combinations is performed as shown in Table 5. Thus, those combinations which apply the light technique are eliminated for attack techniques in which there has recently been an intrusion received. To avoid the excessive use of heavy techniques, the time of 30 min is fixed to take account of recent attacks. In the event that it is unable to complete any combination within the indicated time period, the function indicates this fact in its response and analysis is not performed. The classifier agent should in this case reject the service request since it cannot guarantee its security.

The different techniques that the classifier agent executes once the optimal combination of techniques has been determined include a set of common inputs that are represented by p_c and are defined as follows: $p_c = \{p_1, p_{28}, p_{29}, p_2, p_3, p_4, p_5, p_6, p_{16}, p_{17}, p_{26}, p_{27}\}$. The remaining entries vary according to the techniques used, which is specified for each one. Table 6 shows the information of the different soft computing techniques used for each of the attacks.

The information used by different techniques varies according to the type of attack. Table 7 details the different fields associated with each of the attacks for Heavy techniques. Light techniques only use the fields that refer to the request headers, specifically the following fields $\{h_3, h_4, h_5, h_6\}$.

In addition to the techniques displayed in Table 6, there is a global neural network that contains each of the inputs listed in Table 7 and that is trained for the entire set of cases. This network will be used in those situations where the response time is critical and it is not possible to check each case individually.

Fig. 1 shows the mixture of experts as it is carried out. It is possible to see how input data gathered from the cases are distributed between the different attack classification techniques. Fig. 1 shows only one part of the mixture. The complete description of techniques associated with each attack is shown in Table 6.

At the end of the reuse stage, the optimal output is selected, corresponding to the maximum values provided by each of the experts, so that if any exceeds a given threshold, the service request is considered to be an attack, and classified as such. Keeping in mind that the classification of each attack mechanism is performed by an expert, it is possible to determine the type of attack that was initiated and, additionally, to identify the object at which the attack was targeted (database, parser, etc.).

Once the analysis is complete, if an attack has been detected, the service request is rejected and is not sent to the respective provider. Subsequently, the result of the analysis is evaluated by a human expert through the revise and retain phase, if it is necessary to store the case associated with the request.

The following section describes the different techniques for detecting attacks as listed in Table 6. The description is somewhat general, without specifying the particular inputs associated with any type of attack.

4.2.1. *Decision tree*

This is a knowledge extraction and classification technique widely used in different arenas, from simple statistics to bioinformatics [42]. It involves an advanced technique that has been studied from a variety of perspectives. Among the different types of decision trees are: CLS (Concept Learning System) [43], ID3 (Induction Decision Trees) [44], CART (Classification and Regression Trees) [45], OC1 (Oblique Classier 1) [46], ASSISTANT [47] o C4.5, J48, C5.0/See5 [48]. The J48 algorithm is a non-parametric test that uses extrac-

```
Input  : cases, problem
Output: solution
2.1  switch time do
2.2      case wcet($C_1$) $\leq$ time $<$ wcet($C_2$)
2.3          solution = execute($C_1$)
2.4      end
2.5      case wcet($C_2$) $\leq$ time $<$ wcet($C_3$)
2.6          solution =execute($C_2$)
2.7      end
2.8      ...
2.9      case wcet($C_{n-1}$) $\leq$ time $<$ wcet($Cn$)
2.10         solution =execute($C_{n-1}$)
2.11     end
2.12     case wcet($Cn$) $\leq$ time
2.13         solution =execute($C_n$)
2.14     end
2.15     otherwise
2.16         solution = "Not Enough Time"
2.17     end
2.18 end
2.19 return solution
```

Algorithm 2. adaptProblem function.

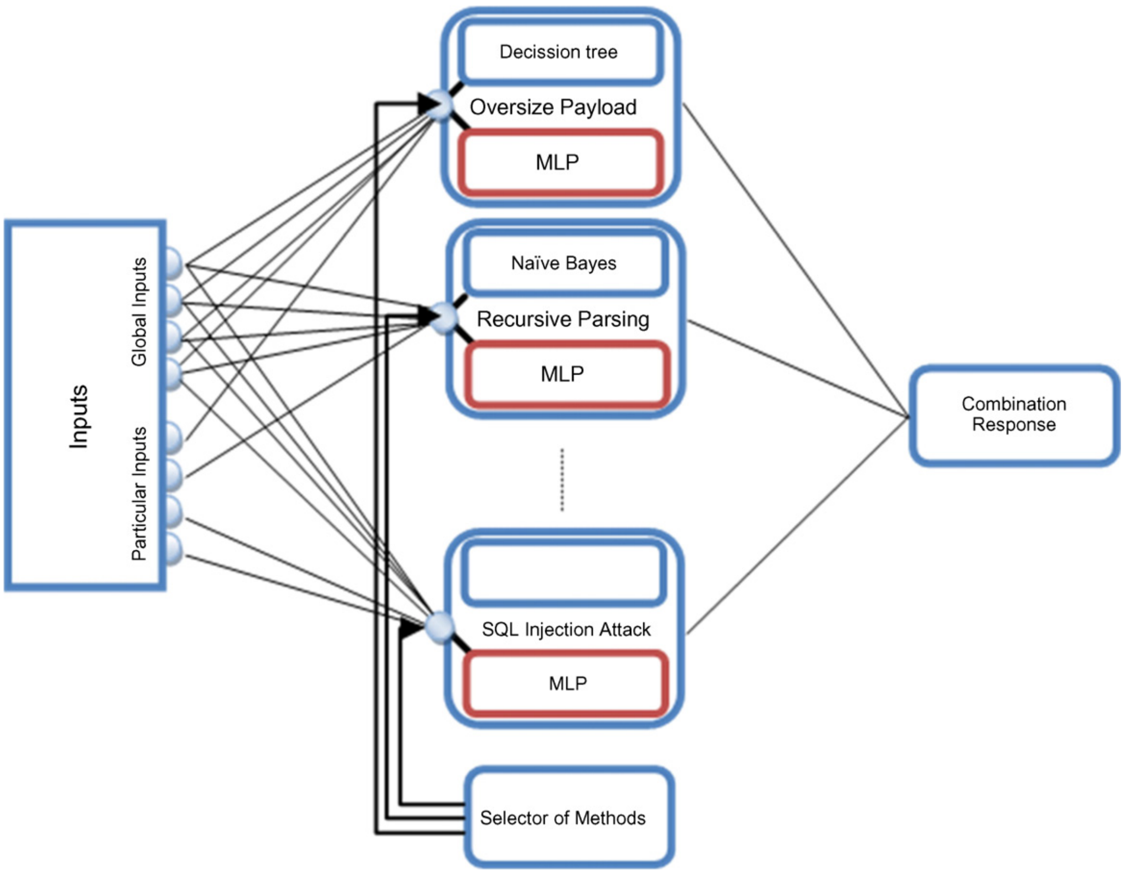


Fig. 1. Multiple method inputs and outputs.

tion rules to explain the classification carried out in the previous steps and to classify new elements. This method makes it possible to generate rules and to extract the most important variables to classify new cases with high performance. Because the results of the different algorithms are very similar, it is only necessary to select one that can work with continual variables, such as CART or J48. We have selected J48 (C,4,5) because it can be used in data mining libraries.

The J48 algorithm attempts to minimize the width of the decision tree using heavy search strategies, which require that any training takes place offline. In summary, the algorithm defines two terms: gain and rate of gain with respect to the information $I(S)$ contained in a node S. Using only the gain criteria, attributes with multiple values are more highly favored given that they can more easily divide the elements into numerous subsets. To avoid the effect of favoring attributes with multiple values, the concept of gain rate is added.

$$I(S) = -\sum_{j=1}^{n} f_j^S \cdot \log(f_j^S) \tag{4}$$

where f_j^S represents the relative frequency of C_j in class S, $f_j^S = n_j^S/N^S$, n_j^S, number of elements in class C_j in S and N^S the total number of elements. This criterion is adjusted for continuous or categorical components.

The earning function is defined as follows:

$$G(S, B) = I(S) - \sum_{i=1}^{t} \frac{|S_i|}{S} I(S_i) \tag{5}$$

B represents the test that separates the modes from S in $S_1 \ldots S_t$ in order to maximize the value of the function $G(S,B)$, $|S_i|$ the number of elements in node S_i.

To avoid favoring the partitions with branches containing few elements, the rate of earnings term is introduced in the following manner:

$$P(S, B) = -\sum_{i=1}^{t} \frac{|S_i|}{S} \log\left(\frac{|S_i|}{S}\right) \tag{6}$$

Finally test B is selected from the previous tests since it maximizes the following criteria:

$$\frac{G(S, B)}{P(S, B)} \tag{7}$$

It is possible to see how processing and calculation time is high and depends on the number of cases, so that for each pair of service-subnet masks, the decision tree is stored in the corresponding memory of rules. The memory of rules is defined by a set of inductive rules defined as follows: $R = \{r_1 \ldots r_l\}$ with $r_i = (l_1 \wedge \cdots \wedge l_m) \rightarrow x_j$ where $l_s = (d_{ts}, o_s, \Re)/d_{ts} \in \{a_i\}$, $o_s \in O$, where a_i is the set of attributes indicated in Table 7, $O = \{=, \neq, >, <, \leq, \geq\}$, $x_j \in X$. The memory of rules is fragmented the same way for each of the services and for each of the network masks and types of attack, so that $R/C_{\cdot h_1 h_2}$ represents the rules associated with the cases belonging to service i and network mask m. The predicted classification for the new case c_{n+1} is carried out from the recovered tree and from the information on the new case. The final value corresponds to the correction rate, the total number of classified nodes divided by the total number of nodes classified by the leaf node associated with the tree. This value is represented by the following function: m_d^i

$$m_d^i\left(\frac{R}{C_{\cdot h_1 h_2}, c_{n+1}}\right) = \frac{n_{td}}{n_t} \tag{8}$$

where i refers to the type of attack, n_{td} refers to the number of type d nodes in the selected leaf note t, and n_t is the number of nodes from leaf node t.

4.2.2. Neural Network

The reasoning memory used by the agent is defined by the following expression: $P = \{p_1 \ldots p_n\}$ and is implemented by means of a MLP (Multilayer Perceptron) neural network. Each P_i is a reasoning memory related to a group of cases according to the service and subnet mask of the client, as denoted by $P_r/C_{\cdot h_1 h_2}$. MLP is the most widely applied and researched artificial neural network (ANN) model. MLP networks implement mappings from input space to output space and are normally applied to supervised learning tasks [49]. Sigmoidal function was selected as the MLP activation function, with a range of values in the interval [0,1]. It is used to detect if a SOAP message is classified as an attack or not. The value 0 represents a legal message (non attack) and 1 a malicious message (attack). The sigmoidal activation function is given by

$$f(x) = \frac{1}{1 + e^{-a}} \tag{9}$$

Entries for the neural network corresponding to the case elements are defined in Table 7. The output corresponds to x^r. Because the neurons exiting from the hidden layer of the neural network contain sigmoidal neurons with values between [0,1], the incoming variables are redefined so that their range falls between [0.2–0.8]. This transformation is necessary because the network does not deal with values that fall outside of this range. The outgoing values are similarly limited to the range of [0.2,0.8] with the value 0.2 corresponding to a non-attack and the value 0.8 corresponding to an attack. Training for the network is carried out by the error Backpropagation Algorithm [50]. The weights and biases for the neurons at the output layer are updated by the following equations:

$$w_{kj}^p(t+1) = w_{kj}^p(t) + \eta(d_k^p - y_k^p)(1 - y_k^p)y_k^p y_j^p + \mu(w_{kj}^p(t) - w_{kj}^p(t-1)) \tag{10}$$

$$\theta_k^p(t+1) = \theta_k^p(t) + \eta(d_k^p - y_k^p)(1 - y_k^p)y_k^p + \mu(\theta_k^p(t) - \theta_k^p(t-1)) \tag{11}$$

The neurons at the intermediate layer are updated by following a procedure similar to the previous case and using the following equations:

$$\begin{aligned} w_{ji}^p(t+1) &= w_{ji}^p(t) + \eta(1 - y_j^p)y_j^p \\ &\times \left(\sum_{k=1}^{M}(d_k^p - y_k^p)(1 - y_k^p)y_k^p w_{kj})x_i^p + \mu(w_{ji}^p(t) - w_{ji}^p(t-1)\right) \end{aligned} \tag{12}$$

$$\begin{aligned} \theta_j^p(t+1) &= \theta_j^p(t) + \eta(1 - y_j^p)y_j^p \\ &\times \left(\sum_{k=1}^{M}(d_k^p - y_k^p)(1 - y_k^p)y_k^p w_{kj}) + \mu(\theta_j^p(t) - \theta_j^p(t-1)\right) \end{aligned} \tag{13}$$

where w_{ji}^p represents the weight that joins neuron j from the intermediate layer with neuron k from the exit layer, t the moment of time and p the pattern in question. d_k^p represents the desired value, y_k^p the value obtained for neuron k from the output layer, y_j^p the value obtained for neuron j from the intermediate layer, η the

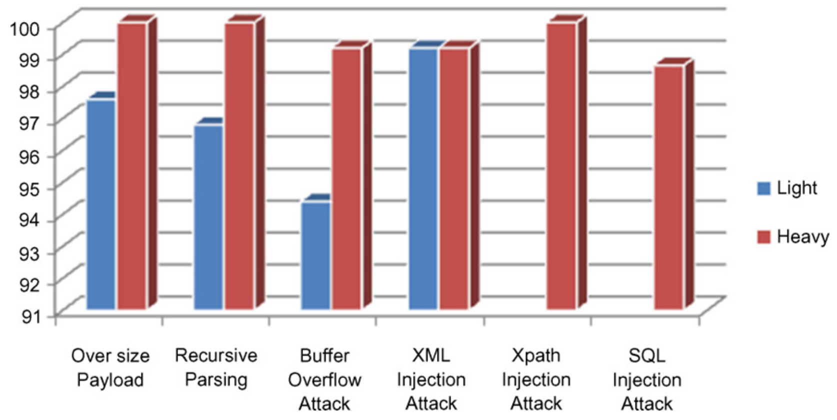


Fig. 2. Percentage of attacks correctly classified.

learning rate and μ the momentum. θ_k^p represents the bia value k from the output layer. The variables for the intermediate layer are defined analogously, keeping in mind that i represents the neuron from the input level, j is the neuron from the intermediate level and M is the number of neurons from the output layer.

When a previously trained network is already available, the message classification process is carried out in the revise phase. If a previously trained network is not available, the training is carried out following the entire procedure and beginning with the cases related to the service and subnet mask, as shown in Eq. (11).

$$\frac{p_r}{C_{\cdot h_1 h_2}} = MLP(C_{\cdot h_1 h_2}) \tag{14}$$

As with the previous technique, the final classification is carried out according to the information from the memory, i.e., the networks and the new case. It involves applying a neural network associated with the case and estimating the output. This is represented in the following manner:

$$m_n^i\left(\frac{p_r}{C_{\cdot h_1 h_2}, c_{n+1}}\right) \tag{15}$$

where i refers to the type of attack. As with the previous case, the neural network previously trained offline is used for the service, type of attack and network mask.

4.2.3. Naive Bayes

In this technique, attacks are divided between the type desired in the study and in the rest of the cases, whether they are attacks or not. The Naive Bayes [51] is then applied. In order to perform an analysis, it is important to consider the type of data, since it is necessary to work with continuous variables or with many categories. As a result it is not possible to apply the Bayes classifier in its original definition since, by so doing, the final probability for each of the classes would be zero, because of the existence of variables with a variety of values.

$$P_a = P(X=a)\prod_{i=1}^{n}\begin{cases} P(A_i = a_i | X = a) & a_i \text{ discrete} \\ P(A_i < a_i | X = a) & a_i \text{ continue}\end{cases} \tag{16}$$

$$P_g = P(C=g)\prod_{i=1}^{n}\begin{cases} P(A_i = a_i | c = g) & a_i \text{ discrete} \\ P(A_i > a_i | c = g) & a_i \text{ continue}\end{cases} \tag{17}$$

where $X=\{a,g\}$ $a=a$ determined type of attack, g = good, u = undefined and a_i corresponds to each of the fields indicated in Table 7 for the studied attack.

The final value obtained by the process is calculated as follows:

$$m_b^i(C_{\cdot im}, c_{n+1}) = \frac{P_a}{P_e} \tag{18}$$

4.2.4. SMO

The Support Vector Machine (SVM) is a supervised learning technique applied to the classification and regression of elements. SVM can be applied in a variety of fields such as chemistry, ambient intelligence, modelling and simulation, and data or text mining. The algorithm represents an extension of the linear models [52]. SVM also allows the separation of element classes which are not linearly separable. For this the space of initial coordinates is mapped in a high dimensionality space through the use of functions. Due to the

Table 8
Description of web services used for the tests.

Input parameter	Type
RequestTreamentPatient: consult a treatment for a patient via Internet	
IdPatient	Int
Start_Time_Treatment	Date
Start_Date_Treatment	Date
End_Time_Treatment	Date
End_Date_Treatment	Date
RequestScheduleDoctor: consult the agenda of a doctor via Internet	
IdDoctor	Int
Date_Schedule	Date
Time_Schedule	Date
RequestAppointment: request an appointment with a doctor via Internet	
IdDoctor	Int
PatientName	String
Date_Appointment	Date
Time_Appointment	Date
Description	String

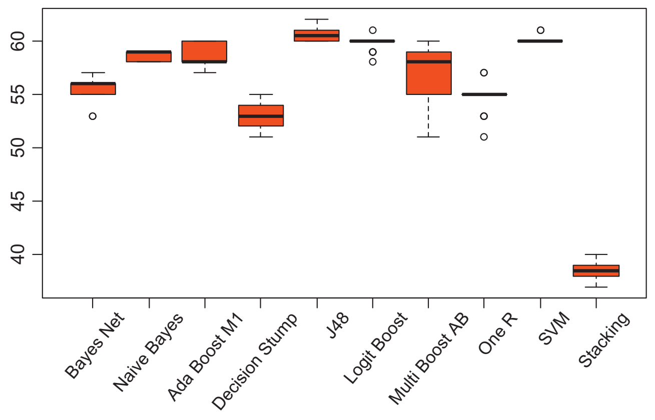


Fig. 3. Boxplots with the number of elements correctly classified.

Table 9
Number of successful classifications obtained for the test and the 5 × 2-Cross-Validation. The last column shows the average number of successful classifications.

	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	S1	S2	Average
BayesNet	56	56	55	56	57	53	56	57	56	55	56	57	56	56	56	56	55	55	55	53	55.32
NaiveBayes	59	59	59	58	59	58	58	59	59	59	59	58	59	59	59	58	58	59	59	58	58.77
AdaBoostM1	58	58	59	57	60	58	58	57	58	58	60	60	58	59	60	58	59	58	60	60	58.66
DecisionStump	52	54	54	52	54	51	55	54	54	52	52	54	52	53	53	52	52	53	53	52	52.97
J48	61	60	62	60	61	60	60	60	61	60	61	60	61	60	61	60	61	60	61	61	60.44
LogitBoost	61	60	60	58	60	60	60	60	59	59	60	60	60	59	60	60	60	60	60	60	59.77
MultiBoostAB	59	59	60	59	54	51	57	55	55	55	59	59	55	54	59	59	59	59	57	57	56.85
OneR	53	57	57	53	55	51	55	55	55	53	55	55	55	55	55	55	55	55	55	55	54.72
SVM	60	61	60	60	60	60	60	60	60	60	60	60	60	60	61	60	60	60	60	60	60.22
Stacking	37	40	40	37	38	39	39	38	39	38	38	39	40	37	37	40	38	39	39	38	38.53

Table 10
Number of successful classifications obtained for the test and 5 × 2-Cross-Validation. The last column shows the average number of successful classifications.

	BayesNet	NaiveBayes	AdaBoostM1	DecisionStump	J48	LogitBoost	MultiBoostAB	OneR	SVM	Stacking
BayesNet		0.0000	0.0000	0.0000	0.0000	0.0000	0.0171	0.0102	0.0000	0.0000
NaiveBayes	0.0000		1.0000	0.0000	0.0000	0.0000	0.0111	0.0000	0.0000	0.0000
AdaBoostM1	0.0000	1.0000		0.0000	0.0000	0.0001	0.0105	0.0000	0.0000	0.0000
DecisionStump	0.0000	0.0000	0.0000		0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
J48	0.0000	0.0000	0.0000	0.0000		0.0002	0.0000	0.0000	0.0086	0.0000
LogitBoost	0.0000	0.0000	0.0001	0.0000	0.0002		0.0001	0.0000	0.0553	0.0000
MultiBoostAB	0.0171	0.0111	0.0105	0.0000	0.0000	0.0001		0.0001	0.0000	0.0000
OneR	0.0102	0.0000	0.0000	0.0000	0.0000	0.0000	0.0001		0.0000	0.0000
SVM	0.0000	0.0000	0.0000	0.0000	0.0086	0.0553	0.0000	0.0000		0.0000
Stacking	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	

fact that the dimensionality of the new space can be very high, it is not feasible to calculate hyperplanes that allow the production of linear separability. For this, a series of non-linear functions called kernels is used.

If we consider a set of patterns marked by $T = \{(x_1,y_1),(x_2,y_2), \ldots,(x_m,y_m)\}$ where x_i is a vector with dimension n, the idea is to convert the elements x_i in a highly dimensional space using the application of a feature function $\Phi(x)$, so that the original set of patterns changes to the following $\Phi(t) = \{(\Phi(x_1),y_1), (\Phi(x_2),y_2), \ldots, (\Phi(x_m),y_m)\}$, according to the $\Phi(x)$ function selected, $\Phi(T)$ could be linearly separated.

The following equation is used to perform the classification (19) [53].

$$\text{class}(x_k) = \text{signe}\left(\sum_{i=1}^{m} \lambda_i y_i \Phi(x_i)\Phi(xk) + b\right) \tag{19}$$

where λ_i is a Lagrange multiplier, y_i output value for the patter x_i, b constant.

As we can see, there is a product $\Phi(x_i)\Phi(x_k)$ that, according to the dimensionality of the new space, can be very costly to calculate. For this reason, it is necessary to select a series of kernel functions that can operate in the original space to perform these calculations without requiring a heavy computational load. Under certain conditions the product in the feature space has a result that is an equivalent kernel in the input space, which is (20)

$$k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \tag{20}$$

For all cases k complying with condition (20), the function is said to be nuclear. The work [53] presents various nuclear functions such as linear, polynomial, Gaussian and exponential, by which there are clearly a variety of nuclear functions that can perform this calculation simply. For example, in the case of the polynomial function, the following exists:

$$k(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) = (1 + x_i x_j)^d \tag{21}$$

Clearly, in order to perform the calculation it is not necessary to consider the new highly dimensional space, so the operations that must be performed to classify individuals $\Phi(x_i)\Phi(x_k)$ are limited to what is indicated on the right of the equation.

To calculate the classifier class (x_k) there are algorithms such as the Sequential Minimal Optimization (SMO) [54] that make is possible to efficiently perform an iterative calculation of the classifier function. Performing an iterative calculation facilitates the temporal bounding of the algorithm since it can temporally bind the solution.

$$m_s^i(C_{\cdot im}, c_{n+1}) = \text{class}(C_{n+1}) \tag{22}$$

Nevertheless, although there is an efficient method of calculation, the classifier is stored in the same manner and is retrieved to perform the classification in the same way as with the previous techniques. The inputs vary according to the attack mechanism being investigated as outlined in Table 7.

Table 11
Number of correct, suspicious or incorrect responses and the average execution time for different combinations.

	Correct	Suspicious	Incorrect	Time (ms)
Single algorithm	1457	34	9	0.563
C_1	1463	14	23	0.611
C_{16}	1484	12	4	1.722

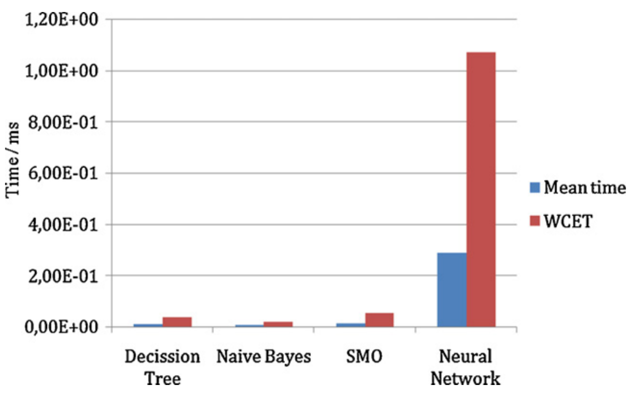


Fig. 4. Average time and Worst Case Estimated Time (WCET).

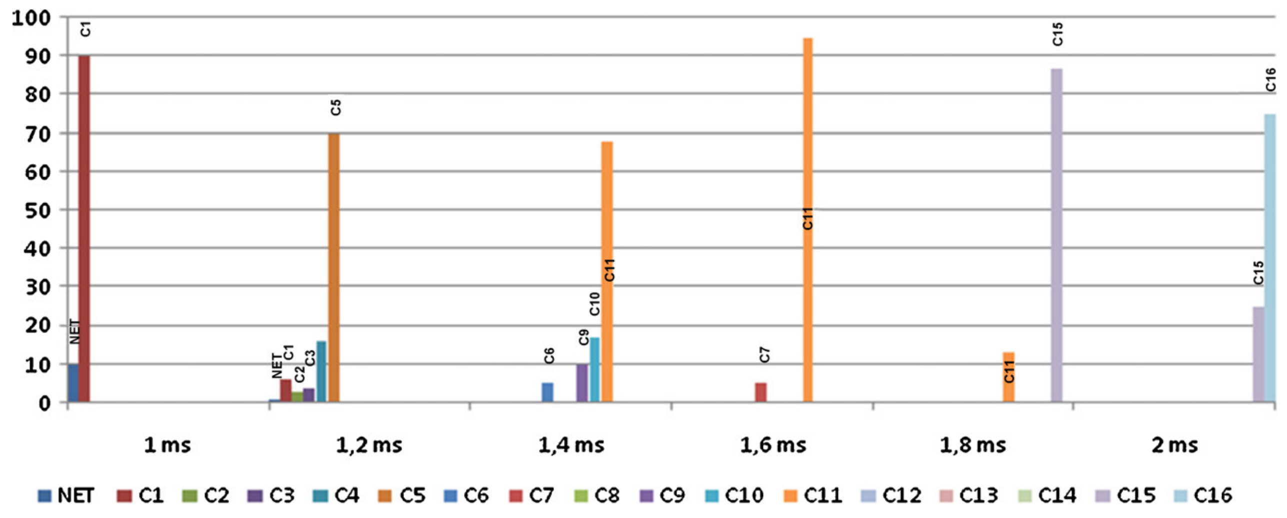


Fig. 5. Queries made for each combination according to time.

5. Validation test: detection of attacks using of our classifier agent

In order to evaluate our prototype, we used a description of three services available from an application currently under development for testing purposes. Table 8 lists a description of these services.

Some technical aspects of the equipment used to conduct the tests will now be provided. These aspects are an influential factor on the results obtained, since the performance of the system is a critical factor when assessing this type of approach. The prototype, and more specifically the classification mechanism, was tested using a standard PC with a 100 Mbps Ethernet network connection. The PC used by the classifiers was an HP Pavilion Intel Core 2 Duo E7200 with 4 GB RAM. In order to initiate requests, we developed a type of requesting agent whose only function was to initiate a series of requests for web services, as described in Table 8. There were a total of three instances of the requesting agent, each of which was located in a different network and sent requests using a different subnet mask.

Both the requesting agent and the real time classifier agent are executed on the jART platform [55], which is a multi-agent platform system specially designed to work in real time environments. The use of this platform is necessary to guarantee the execution of tasks with temporal restrictions that must be carried out by the real time classifier agent. At the same time, both the agent and the platform work in a new real time operating system.

6. Results and conclusion

The availability of web services has been threatened by a variety of attack mechanisms that can lead to a denial of service attack that, in the worst case scenario, can cause authorized users to lose access to the services and available resources.

This article has presented a novel proposal for detecting and blocking attack mechanisms in web service requests. The proposal includes different soft computing classification techniques that are integrated in a real-time case-based reasoning system, in such a way that the system guarantees robustness and low solution cost. The article proposes a new vision in which each attack mechanism is individually analyzed. This makes it possible to divide the general problem into smaller sub-problems, which facilitates the application of classification techniques according to the characteristics associated with each attack mechanism. It also makes it possible to obtain better classification results with regard to both the effectiveness of the classification process and the response time, since all classification mechanism tasks are temporally bounded.

In order to validate the initial prototype, we proposed a benchmark case study that used description 3 from the web services, designed especially for testing. The service requests were initiated from an agent specifically developed for that task and they were classified by a classifier agent. The following section describes in greater detail the tests that were carried out.

The description for each of these services was used to build a set of tests to validate the effectiveness of our Classifier agent. The first set of tests was developed to determine whether the use of a mixture of experts was able to improve the detection and subsequent classification of attack mechanisms. The second set of tests focused on demonstrating the convenience of using our proposal

Table 12
Comparison of current approaches vs. TB-CBR.

	Nedgty [11]	XML Firewall [13]	CheckWay gateway [58]	IAPF [34]	ID/IP framework [15]	Twofold mechanism [16]	DDoS and XDoS defense system [17]	SOTA and XDetector [18]	TB-CBR
Learning ability	No	No	No	No	Yes	No	No	Yes	Yes
Adaptive ability	No	No	No	No	Yes	No	No	No	Yes
Positive and negative False	–	No	–	–	–	–	–	Yes	Yes
Time response	–	–	Nearly real time	–	–	Nearly real time	Nearly real time	–	Real time

in environments where it is necessary to consider certain temporal restrictions associated to the process of analysis and classification.

Prior to initiating the tests, the attack classification mechanisms were analyzed for each use of a light or heavy technique. The analysis demonstrated that the use of heavy techniques provided a better classification, but with a greater temporal cost. Although this result was expected and logical, the tests were performed in order to empirically prove the execution times in the worst case for each of the different techniques. The time was used to determine which set of techniques should be applied so as to not exceed the deadline, which was not exceeded in any of the executions.

The first test compares a series of results obtained for different attacks and techniques. In order to perform these tests, 1500 requests were generated, of which 750 were legal and 750 illegal, distributed in groups of 125, 125, 125, 125, 100 and 150 for each type of attack: oversize payload, recursive parsing, buffer overflow attack, XML injection attack, Xpath injection attack and SQL injection attack, respectively. The techniques indicated in Table 6 were applied to each attack using the Leave-one-out Cross-Validation soft computing technique [59,60]. Fig. 2 lists the percentage of attacks correctly classified for light and heavy techniques. It is clear that light techniques have a lower rate of correctness than heavy techniques when dealing with the different attacks for which they are available.

To compare the different soft computing techniques used in the light classifier, it was necessary to train the system and carry out a cross validation following the Dietterich's 5 × 2-Cross-Validation Paired *t*-Test algorithm [61,62] and instead of the Leave-one-out technique. A detailed analysis was carried out for the different soft computing methods in a similar way to the oversize payload, recursive parsing attack shown in Table 9. 125 requests were selected and the 5 × 2-Cross-Validation was carried out. Table 9 shows the number of successful classifications for the test performed for the different soft computing techniques taken into account in this experiment. These techniques are those based on decision trees, as J48, decision Stump, OneR decision rules, statistic techniques to provide probabilistic values that can be considered as fuzzy values, as Naive Bayes and Bayesian network, and the mixture of multiple classifiers based on boosting or Stacking.

From the results shown in Table 9 it is possible to analyze the relevance of the differences using the statistic techniques presented in Dietterich's 5 × 2-Cross-Validation Paired *t*-Test algorithm [62]. Once the Paired *t*-Test was applied, it was possible to obtain the results shown in Table 10. The results presented in Table 10 indicate that the j48 method provides better results than the rest of the soft computing methods for a significance level of 0.05, since the *p* value obtained is lower than for the rest of the methods.

Additionally, if the statistical dispersion of the data is analyzed by means of a box plot in Fig. 3, then it is possible to observe how the average rate for succesful classifications using the J48 method is better than those obtained for the rest of the soft computing methods. This fact, together with fact that the average rate for the J48 method is higher than the rest of the methods, allows us to conclude that the J48 soft computing method provides the best results.

Once the rate of correctness was analyzed, we proceeded to analyze the results with regard to execution time. A total of 1500 available queries were used to analyze the execution time. We replicated and analyzed 30,000 executions to calculate the average execution time. The final result is shown in Fig. 4, which shows that execution time for the neural network to perform its estimates is much greater than the other techniques.

Table 11 shows the results obtained. As can be seen, the single algorithm based on a single neural network carries out the estimation of attack using all input parameters. The neural network generates the worst estimated result but it is the fastest.

In order to analyze the final efficiency of the system, we forced it to function under various conditions so that we could determine the variation in the correctness rate and the average execution time for the 1500 requests. To accomplish this, a variable deadline takes a value from one that is equal to the time that the classifier agent needs to perform an analysis using the C_1 combination (the fastest), and another slightly greater value that the agent needs to complete the analysis indicated by the combination C_{16}. Table 11 shows the results obtained. Clearly, the unique algorithm created by a unique neural network that can produce an attack estimate under all the worst case input parameters is the estimate for the worst case, but also the fastest estimate. The fact that it is the fastest is that combination C_1 also includes two networks, albeit more simple, that can estimate two of the attacks. With regards to the correctness rate, the most efficient combination is C_{16}, which is the heaviest.

For the second test a set of 100 queries were selected and then classified according to different pre-determined deadlines. The results are shown in Fig. 4. For the deadlines with a low value, the fastest combinations were primarily used, while deadlines with a high value used the heaviest combinations. Fig. 5 includes a NET (Not Enough Time) column. This column represents the number of queries that were unable to be analyzed within the given time constraints. The number of NET requests decreases as the deadline increases since it is possible to apply light techniques for those cases where it is not possible to apply the heavy techniques, which have a greater computational cost.

In addition to the techniques presented in this article, there are others that can perform similar classifications such as CART [56,57], which can also be used in the case study. Nevertheless, the primary objective of this study is to integrate the most well-known classification techniques in a real time environment, not to carry out an exhaustive analysis of existing techniques. The analysis could have been completed in a less temporally restrictive manner, but there would be no guarantee of complying with all the temporal restrictions. However, similar classification results would be obtained.

Finally, Table 12 presents a theoretical comparison of TB-CBR with current approaches aimed at detecting DoS attacks in web services environments. Those parameters that could not be evaluated are marked with a hyphen.

As can be seen in the comparative table, TB-CBR provides new important capacities such as a learning capacity and also adaptability which provide our new approach with incremental learning and a flexibility to adapt to attack techniques. Additionally, TB-CBR is one of the only approaches that aims to work in circumstances where response time is critical. If it is true that the use of a CBR that has been temporally bounded, we limit the time for obtaining the optimum analysis, although because of the need to complete the analysis before a given time for this to be valid, we are obliged to limit the deliberation process of the CBR system. Even so, using the TB-CBR system, an algorithm with anytime characteristics, we are guaranteed that the obtained result is the optimum attending to the time available to carry out the analysis.

The results are promising and allow us to conclude that our approach can be considered as a solid alternative to prevent and detect DoS attacks in web service environments. However, there is still much work to be done, especially with regard to checking the validity of our approach in heterogeneous real environments. These are our next challenges.

Acknowledgements

This work has been partially supported by the JCYL-2002-05 Project, SA071A08 Spanish Project, the Spanish government (TIN2009-13839-C03-01), FEDER and CONSOLIDER-INGENIO (2010 CSD2007-00022), the Generalitat Valenciana (PROME-

TEO/2008/051) and the Professional Excellence Program 2006-2010 IFARHU-SENACYT-Panama.

References

[1] M. Stamp, Information Security: Principles and Practice, Wiley InterScience, 2006.
[2] H.F. Tipton, M. Krause, Information Security Management Handbook, sixth edition, Auerbach Publications, Boston, MA, USA, 2007.
[3] A. Nadalin, C. Kaler, R. Monzillo, P. Hallam-Baker, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), 2006.
[4] G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, C. Kaler, H. Maruyama, M. McIntosh, A. Nadalin, N. Nagaratnam, R. Philpott, H. Prafullchandra, J. Shewchuk, D. Walter, R. Zolfonoon, Web services security policy language Version 1.0 (WS-SecurityPolicy), 2005.
[5] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della, B. Dixon, Web Services Trust Language (WS-Trust), 2004.
[6] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, Web Services Secure Conversation Language (WS-Secure Conversation) Version 1.1, 2004.
[7] N. Gruschka, M. Jensen, N. Luttenberger, A stateful web service firewall for BPEL, in: IEEE International Conference on Web Services, 2007, pp. 142–149.
[8] A. Householder, A. Manion, L. Pesante, G.M. Weaver, Managing the Threat of Denial-of-Service Attacks, CERT® Coordination Center – Carnegie Mellon University, 2001.
[9] C.L. Schuba, I.V. Krsul, M.G. Kuhn, E.H. spafford, A. Sundaram, D. Zamboni, Analysis of a denial of service attack on TCP, in: Proceedings of the IEEE Symposium on Security and Privacy (SP'97), IEEE Computer Society, Washington, DC, USA, 1997, p. 208.
[10] E.G. Im, Y.H. Song, An adaptive approach to handle DoS attack for web services, in: S.B. Heidelberg (Ed.), Intelligence and Security Informatics, 2005, pp. 634–635.
[11] R. Bebawy, H. Sabry, S. El-Kassas, Y. Hanna, Y. Youssef, Nedgty. Web services firewall, in: IEEE International Conference on Web Services (ICWS'05), Orlando, Florida, 2005, pp. 597–601.
[12] J. Wang, Defending against denial of web services using sessions, in: IEEE/IST Workshop on: Monitoring, Attacking Detection and Mitigation, 2006.
[13] Y.-S. Loh, W.-C. Yau, C.-T. Wong, W.-C. Ho, Design and implementation of an XML Firewall, in: International Conference on Computational Intelligence and Security, 2006, pp. 1147–1150.
[14] S. Padmanabhuni, V. Singh, K.M.S. Kumar, A. Chatterjee, Preventing service oriented denial of service (PreSODoS): a proposed approach, in: IEEE International Conference on Web Services (ICWS'06), IEEE Computer Society, Washington, DC, USA, 2006, pp. 577–584.
[15] C.G. Yee, W.H. Shin, G.S.V.R.K. Rao, An adaptive intrusion detection and prevention (ID/IP) framework for web services, in: International Conference on Convergence Information Technology (ICCIT'07), IEEE Computer Society, Washington, DC, USA, 2007, pp. 528–534.
[16] M. Srivatsa, A. Iyengar, J. Yin, L. Liu, Mitigating application-level denial of service attacks on web servers: a client-transparent approach, ACM Transactions on Web 2 (2008) 1–49.
[17] X. Ye, Countering DDoS and XDoS attacks against web services, in: IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, IEEE Computer Society, Washington, DC, USA, 2008, pp. 346–352.
[18] A. Chonka, W. Zhou, Y. Xiang, Defending grid web services from XDoS attacks by SOTA, in: EEE International Conference on Pervasive Computing and Communications, IEEE Computer Society, Los Alamitos, CA, USA, 2009, pp. 1–6.
[19] C. Carrascosa, A. Terrasa, F.A. García, A. Espinosa, V.J. Botti, A Meta-Reasoning Model for Hard Real-Time Agents, CAEPIA, 2005, pp. 42–51.
[20] D.M. Surka, M.C. Brito, C.G. Harvey, The real-time object agent software architecture for distributed satellite systems, in: Proceedings of the IEEE Aerospace Conference, 2001, pp. 2731–2741.
[21] K. Prouskas, J. Pitt, Towards a real-time architecture for time-aware agents, in: First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02), ACM, New York, NY, USA, 2002, pp. 92–93.
[22] A. Ram, J.C. Santamaría, Continuous case-based reasoning, Artificial Intelligence 90 (1997) 25–77.
[23] S. Ontañon, K. Mishra, N. Sugandh, A. Ram, On-line case-based planning, Computational Intelligence 26 (2010) 84–119.
[24] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, G.E. Hinton, Adaptive mixtures of local experts, Neural Computation 3 (1991) 79–87.
[25] A.J. Garvey, V.R. Lesser, Design-to-time real-time scheduling, IEEE Transactions on Systems, Man and Cybernetics 23 (1993) 1491–1502.
[26] A. Subasi, EEG signal classification using wavelet feature extraction and a mixture of expert model, Expert Systems with Applications 32 (2007) 1084–1093.
[27] C.I. Pinzón, Y. De Paz, J. Bajo, A multiagent based strategy for detecting attacks in databases in a distributed mode, in: J.M. Corchado, S. Rodríguez, J. Llinas, J.M. Molina (Eds.), International Symposium on Distributed Computing and Artificial Intelligence (DCAI'8), Springer, Berlin/Heidelberg/Salamanca, Spain, 2008, pp. 180–188.
[28] J. Bajo, J.M. Corchado, C. Pinzón, Y.D. Paz, B. Pérez-Lancho, SCMAS. A distributed hierarchical multi-agent architecture for blocking attacks to databases, International Journal of Innovative Computing, Information and Control (2008).
[29] K. Zhao, K. Yang, M. Zhang, J. Wang, L. Hu, Denial of service attack simulation based-on CASL, in: IEEE International Workshop on Anti-counterfeiting, Security, Identification, 2007, pp. 266–269.
[30] E. Pulier, H. Taylor, Understanding Enterprise SOA, Manning Publications Co., Greenwich, CT, USA, 2005.
[31] E. Cerami, Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, first edition, O'Reilly & Associates, 2002.
[32] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H.F. Nielsen, A. Karmarkar, Y. Lafon, W3C Recommendation: SOAP Version 1.2 Part 2: Adjuncts, second edition, 2007.
[33] E. Moradian, A. Håkansson, Possible attacks on XML web services, International Journal of Computer Science and Network Security 6 (2006) 154–170.
[34] N. Sidharth, J. Liu, A framework for enhancing web services security, in: 31st Annual International Computer Software and Applications Conference (COMPSAC'07), IEEE Computer Society, Washington, DC, USA, 2007, pp. 23–30.
[35] V. Julián, V. Botti, Developing real-time multi-agent systems, Integrated Computer-Aided Engineering 11 (2004) 135–149.
[36] J.A. Stankovic, Misconceptions about real-time computing: a serious problem for next-generation systems, IEEE Computer 21 (1988) 10–19.
[37] A. Garvey, V. Lesser, A survey of research in deliberative real-time artificial intelligence, Real-Time Systems 6 (1994) 317–347.
[38] V.J. Botti, C. Carrascosa, V. Julián, J. Soler, Modelling agents in hard real-time environments, in: 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Springer-Verlag, London, UK, 1999, pp. 63–76.
[39] A. Martens, A.M. Uhrmacher, Adaptative tutoring processes and mental plans, in: S.A.a.G.G. Cerri, F. Paraguaçu (Eds.), Proceedings of Intelligent Tutoring Systems—ITS, Springer, Berlin, 2002, pp. 71–80.
[40] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations, and system approaches, AI Communications 7 (1994) 39–59.
[41] T. Dean, M.S. Boddy, An analysis of time-dependent planning, in: 7th National Conference on Artificial Intelligence, 1988, pp. 49–54.
[42] J.M. Corchado, J.F. De Paz, S. Rodríguez, J. Bajo, Model of experts for decision support in the diagnosis of leukemia patients, Artificial Intelligence in Medicine 46 (2009) 179–200.
[43] E.B. Hunt, J. Marin, P.J. Stone, Experiments in Induction, Academic Press, New York, 1966.
[44] J.R. Quinlan, Discovering rules by induction from large collections of examples, in: E.U. Press (Ed.), Expert Systems in the Micro-electronic Age, Edinburgh, Scotland, 1979, pp. 168–201.
[45] L. Breiman, J. Friedman, C.J. Stone, R.A. Olshen, Classification and Regression Trees, Wadsworth and Brooks, Belmont, CA/Monterey, CA, USA, 1984.
[46] N.N. Murthy, B.M. Mehtre, K.P.R. Rao, G.S.R. Ramam, P.K.B. Harigopal, K.S. Babu, Technologies for E-Commerce: An Overview, 2001. Available from: http://www.cmcltd.com/brochures/products_solutions/techecom-paper.pdf.
[47] B. Cestnik, I. Kononenko, I. Bratko, Assistant 86: a knowledge-elicitation tool for sophisticated users, in: Progress in Machine Learning-2nd European Working Session on Learning (EWSL 87), Bled, Yogoslavia, 1987, pp. 31–45.
[48] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Francisco, CA, USA, 1993.
[49] M. Gallagher, T. Downs, Visualization of learning in multilayer perceptron networks using principal component analysis, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics 33 (2003) 28–34.
[50] Y. LeCun, L. Bottou, G.B. Orr, K.R. Müller, Efficient BackProp, in: Neural Networks: Tricks of the Trade, Springer, Berlin/Heidelberg, 1998, p. 546.
[51] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, John Willey and Sons, New York, 1973.
[52] V. Vapnik, A. Lerner, Pattern Recognition Using Generalized Portrait Method, 1963, pp. 774–780.
[53] O. Ivanciuc, in: K.B. Lipkowitz, T.R. Cundari (Eds.), Applications of Support Vector Machines in Chemistry, John Wiley & Sons, 2007, pp. 291–400.
[54] J.C. Platt, Fast Training of Support Vector Machines Using Sequential Minimal Optimization (1999), pp. 185–208.
[55] M. Navarro, V. Julian, J. Soler, V. Botti, jART: a real-time multi-agent platform with RT-Java, in: 3th International Workshop on Practical Applications of Agents and Multi-Agent Systems (IWPAAMS'2004), Universidad de Burgos, 2004, pp. 73–82.
[56] H.R. Bittencourt, R.T. Clarke, Use of classification and regression trees (CART) to classify remotely-sensed digital images, in: Proceedings IEEE International Geoscience and Remote Sensing Symposium (IGARSS'03), 2003, pp. 3751–3753.
[57] W.W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 115–123.
[58] N. Gruschka, N. Luttenberger, Protecting web services from DoS attacks by SOAP message validation, in: Security and Privacy in Dynamic Environments, Springer, Boston, 2006, pp. 171–182.
[59] G. Cawley, N. Talbot, Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers, Pattern Recognition 36 (11) (2003) 2585–2592.
[60] H. Wang, G. Li, E. Li, comparative study of boundary-based intelligent sampling approaches for nonlinear optimization, Applied Soft Computing (2011) doi:10.1016/j.asoc.2010.08.002.
[61] E. Mężyk, O. Unold, Mining fuzzy rules using an artificial immune system with fuzzy partition learning, Applied Soft Computing (2011) doi:10.1016/j.asoc.2010.06.012.
[62] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, Neural Computation (1998) 1895–1923.

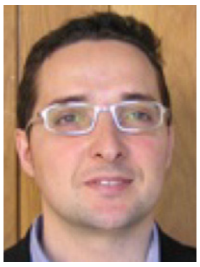
Cristian I. Pinzón (PhD student) is currently completing a PhD in Computer Science at the University of Salamanca (Spain) and obtained a Master's degree in Intelligent Systems in the same university in 2007. He obtained a Bachelor in Computer Sciences Degree at the Technological University of Panama in 2003 and a Technical Engineering in Systems Computer Sciences Degree at the same university in 2000. He has been co-author of published papers in several journals. His studies are supported by the Professional Excellence Program 2006–2010 IFARHU-SENACYT-Panama.

Juan Francisco De Paz (PhD student) is currently completing his studies of PhD in Computer Science at the University of Salamanca (Spain). He is assistant professor at the University of Salamanca. He obtained a Technical Engineering in Systems Computer Sciences Degree in 2003, an Engineering in Computer Sciences Degree in 2005 at the University of Salamanca and a Statistics Degree in 2007 in the same university. He has been co-author of published papers in several journals.

Martí Navarro (PhD student) is a researcher at the GTI-IA Research Group of the Valencia University of Technology, Spain. His research interests include multi-agent systems, agent architectures, multiagent system methodologies and real-time agents. Contact: Departamento de Sistemas Informáticos y Computación, Univ. Politécnica de Valencia, Camino de la Vera S/N, 46022, Valencia, Spain.

Javier Bajo (PhD) received a PhD in Computer Science and Artificial Intelligence from the University of Salamanca in 2007. At present he is associate professor at the Pontifical University of Salamanca (Spain). He obtained an Information Technology degree at the University of Valladolid (Spain) in 2001 and an Engineering in Computer Sciences degree at the Pontifical University of Salamanca in 2003. He has been member of the organising and scientific committee of several international symposiums such as CAEPIA, IDEAL, and HAIS. and is co-author more than 100 papers published in recognized journals, workshops and symposiums.

Vicente Julián (PhD) is an associate professor at the Department of Information Systems and Computation at the Valencia University of Technology and a researcher at the GTI-IA Research Group of the Valencia University of Technology, Spain. His research interests include multi-agent systems, agent architectures, multiagent system methodologies and real-time agents. He received his PhD in Computer Science from the Valencia University of Technology, Spain. Contact: Departamento de Sistemas Informáticos y Computación, Univ. Politécnica de Valencia, Camino de la Vera S/N, 46022, Valencia, Spain.



Juan M. Corchado (PhD) received a PhD in Computer Science from the University of Salamanca in 1998 and a PhD in Artificial Intelligence (AI) from the University of Paisley, Glasgow (UK) in 2000. At present he is dean at the Faculty of Computer Sciences, associate professor and director of the Intelligent Information System Group (http://bisite.usal.es) and director of the MSc programs in Computer Science at the University of Salamanca (Spain). Previously he was sub-director of the Computer Science School at the University of Vigo (Spain, 1999–2000) and researcher at the University of Paisley (UK, 1995–1998). He has been a research collaborator with the Plymouth Marine Laboratory (UK) since 1993. He has led several Artificial Intelligence research projects sponsored by Spanish and European public and private institutions and has supervised seven PhD students. He is the co-author of over 130 books, book chapters, journal papers, technical reports, etc. published by organisations such as Elsevier, IEEE, IEE, ACM, AAAI, Springer Verlag, and Morgan Kaufmann. Most of these present practical and theoretical achievements of hybrid AI and distributed systems. He has been president of the organising and scientific committee of several international symposiums.