Temporal Bounded Reasoning for Context-based Information Fusion in DoS attack Detection

Cristian I. Pinzon
Universidad Tecnológica de Panamá
Campus «Dr. Víctor Levi Sasso» Alfaro, Panamá
cristian_ivanp@usal.es

Martí Navarro
DSIC
Universidad Politécnica de Valencia
Valencia, Spain
mnavarro@dsic.upv.es

Juan F. De Paz
Departamento Informática y Automática Universidad de Salamanca
Salamanca, Spain
fcofds@usal.es

Carolina Zato
Departamento Informática y Automática
Universidad de Salamanca
Salamanca, Spain
carol_zato@usal.es

Javier Bajo
Universidad Pontificia de Salamanca
Salamanca, Spain
jbajope@upsa.es

Vicente Julián
DSIC
Universidad Politécnica de Valencia
Valencia, Spain
vinglada@dsic.upv.es

Abstract – Security is one of the main aspects to take into account when designing and developing web services. A meaning number of mechanisms of attack can lead to a web service system crash. As a result, the web service cannot allow the access to authorized users. This type of attacks is so-called as denial of service attack (DoS) which affects the availability of the services and resources. This paper presents an approach to detect and classify DoS attacks, based on Case-based Reasoning methodology where heterogeneous data are merged in order to obtain a set of cases that reflect previous analysis. The proposal has two peculiarities: (i) when a web service is attacked, this possible attack is studied by different analysers, distributing previous experiences among the analysers, in order to classify attacks in a better way; and (ii) the analysis process against attacks is temporal bounded allowing its use in environments with temporal constraints.

Keywords: CBR, temporal-bounded process, mixture of experts.

1 Introduction

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* [1], *WS-SecurityPolicy* [2], *WS-Trust* [3], *WS-SecureConversation* [4], etc.) only consider the integrity and confidentiality of messages [5] without much consideration for availability.

Recently the availability of web services has been threatened by a well known and studied type of attack known as denial of service (DoS) [6] [7] [5]. 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 that are made within a short period of time [8]. This type of attack results in the interruption of access to the server by authorized users. However, when the problem of DoS attacks is produced in a web services environment, the risk of a DoS attack being carried out increases considerably. Taking into account the fact that web services are grounded in a series of known standards [9] [10] [11], including HTTP, which is the most common means of transporting messages, and the 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.

The DoS attack mechanisms at the 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 attacks on web services was presented by Moradian and Håkansson [15], and includes attack mechanisms that can affect the availability of web services. Table 1 presents the DoS attack mechanisms analyzed within this study.

Attack Mechanism	Description
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.
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.
Schema Poisoning	An attacker compromises XML schema and replaces it with similar, but modified one.
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.
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.
SQL Injection	An attacker inserts and executes malicious SQL statements into XML
XPath Injection	An attacker forms SQL-like queries on an XML document using XPath to extract an XML database.
Replay Attacks	To overload Web Service an attacker steals messages and sends them repeatedly
XML Denial of Service attack	An attacker trying to prevent legitimate users from accessing a service by flooding the service with thousands of requests

Table 1. Types of attacks, target of attack and damage level

There are several initiatives within this field: [16] [17] [18] [5] [19] [20] [21] [22] [23]. However, the main disadvantage common to each of these approaches is their low capacity to adapt themselves to the changes in the patterns, which reduces the effectiveness of these methods when slight variations in the behaviours of the known attacks occur or when new attacks appear. Moreover, most of the existing approaches are based on a centralized perspective. Because of this and the focus on performance aspects, centralized approaches can become a bottleneck when security is broken, causing a reduction of the overall performance of the application. Finally, none of these approaches considers the limitations or restrictions in the response time being this aspect "critical" in most security systems. Within this context, we consider response time as the time available to complete an analysis in order to be considered as valid.

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. To do this, we employ real-time agents [24] that allow us a greater control over existing temporal restrictions in the analysis tasks in order to meet their deadlines. The internal structure of the real-time agent is based on the Case-Base Reasoning (CBR) model, with the main difference being that the different CBR phases are time bounded, thus enabling its use in real time. Additionally, the adaptation phase in the CBR system that is 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 the complicated classification task into a series of simple subtasks, so that the fusion of the solutions given by the sub tasks generates the final solution. To do it, a context-based information fusion and the concept of mixture of experts are united to achieve a correct distribution of the data stored among the different analyzers. The concept of mixture of experts was first proposed by [12]. 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 carrying out the classification, a set of experts is selected to perform the different analyses. The experts are selected with a multiple method model [13]. Finally the different selected experts generate the predictions and the outputs are fused to generate a new unique result [14].

The rest of the paper is structured as follows: section 2 shows a general view of the real-time agent as classifier agent focusing on the temporal bounded CBR used in the deliberative mechanism. Section 3 explains the data used in the construction of cases in the CBR and how these data are distributed among the different methods using the mixture of experts. Section 4 describes a set of tests to evaluate our proposal. Finally, final results and conclusions are presented in section 5.

2 Real-Time agent to DoS analysis

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 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 [24]. 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 [25].

The main problem in the architecture of a Real Time Agent (RTA) is with the deliberation process. This process probably uses Artificial Intelligence (AI) techniques as problem-solving methods to compute more intelligent actions. If this is the case, it is difficult to extract 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 [26] [27].
Therefore, it would be interesting to integrate complex deliberative processes for decision-making in real-time agents 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 behaviour of the environment. Intelligent agents may use a lot of reasoning mechanisms to achieve these capabilities, including planning techniques [28] or Case-Based Reasoning (CBR) techniques [29].

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, the CBR phases must be temporal bounded to ensure that solutions are produced on time, giving the system a temporal bounded deliberative case-based behaviour.

With this in mind, the Temporal Bounded CBR (TB-CBR) [37] was designed. This design consists basically in a modification of the classic CBR cycle, adapting it so that it can be applied in real-time domains. In the TB-CBR we can classify the four phases of a CBR in: 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 [30], where the process is iterative and each iteration is time-bounded and may improve the final response.

Therefore, when extracting and analyzing data from the cases is necessary to take into account that this process must be temporal controlled. It is mandatory that the execution time of any needed operation with the data must be known in an off-line way.

For this reason, the design decision about the data structure of the case-base, number of cases in the case-base and the different algorithms that implement each CBR phase are important factors for determining the execution time of the CBR cycle. Thus, a maximum number of cases in the case-base must be defined by the designer. Note that, usually, the temporal cost of the algorithms that implement these phases depends on this number. For instance, let us assume that the designer chooses a hash table as a data structure for the case-base. This table is a data structure that associates keys to specific values. Search is the main operation that it supports in an efficient way: it allows access to elements (e.g. phone and address) by using a hash function to transform a generated key (e.g. owner name or account) to a hash number that is used to locate the desired value. The average time to make searches in hash tables is constant and defined as O(n) in the worst case. Therefore, if the cases are stored as entries in a hash table the maximum time to look for a case depends on the number of cases in the table. Similarly, if the case-base is structured as an auto-balanced binary tree the search time in the case-base in the worst case would be O(log n). In any case, the retrieval and retention time can be reduced by using an indexing algorithm. These algorithms organize the case-base by selecting a specific feature (or set of features) from the cases, grouping together those cases that share the same values for these features. This reduces the cost of the search for similar cases (for retrieval or previous to the introduction of new cases in the case-base) to a specific set of cases with the same index as the current case [31] [32] [33].

The following section presents the necessary dates in the case-base and the TB-CBR to perform the security analysis for web service requests.

3 TB-CBR adapted for DoS attacks detection

This section presents the TB-CBR, mentioned in the previous section. This TB-CBR will be incorporate to an agent as a reasoning engine. In the TB-CBR used to DoS attacks detection, the learning phase is eliminated since it is now performed by human experts. The TB-CBR utilizes a global case base, which avoids any duplication of the content of any information compiled from the cases or any information contained in the results of the analysis. Tables 2, 3 and 4 show the structure of the cases. Table 2 shows the fields recovered from the analysis of the service request headers. Table 3 shows the fields associated with the analysis of the service requests that were obtained after the analysis performed by the parser application.

	Fields	Type	
Number of header elements	NumberHeaderElement (Int)		p1
Number of elements in the body	NElementsBody (Int)	String	p2
Greatest value associated to the nesting elements	NestingDepthElements(Int)	Int	p3
Greatest value associated to the repeated tag within the body	NXMLTagRepeated (Int)	Int	p4
Greatest value associated with the leaf nodes among the declared parents	NLeafNodesBody (Int)	Int	p5
Greatest value of the associated attributes among the declared elements	NAttributesDeclared (Int)	Int	p6
Type of SQL command	Command_Type		p7
Number of times that the AND operator appears in the string	Number_And		p8
Number of times that the OR operator appears in the string	Number_Or		p9
Number of times the Group By function appears	Number_GroupBy		p10
Number of times that the Order By function appears	Number_OrderBy		p11

Number of times that the Having function appears	Number_Having	p12
Number of Literals declared in the string	Number_Literals	p13
Number of times that the Literal Operator Literal expression appears	Number_LOL	p14
Length of the SQL string	Length_SQL_String	p15
Greatest value associated with the length of the string among the elements or attributes within the body	LengthStringValueBody (Int)	p16
Total number of incidences during the parsing process	TotalNumberIncidenceParsing(Int)	p17
Reference to an external entity	URIExternalReference (Boolean)	P18
Number of variables declared in hte XPath expression	XPathVariablesDeclared (Int)	P19
Number of elements affected in the consulted node	XpathNumberElementAffected(Int)	p20
Number of literals declared in the XQuery Statement	XPathNumberLiterals Declared	p21
Number of times the And operator appears in the XQuery Statement	XPathNumberAndOperator	p22
Número de veces que aparece el operador Or en la XQuery Statement Number of times the Or operator appears in the XQuery Statement	XPathNumberOrOperator	p23
Number of functions declared in the XQuery Statement	XPathNumberFunctionDeclared	p24
Lentgh of the XQuery Statement in a SOAP message	XPathLenghtStatement	p25
Cost of processing time (CPU)	CPUTimeParsing (Int)	p26
Cost of memory size (KB)	SizeKbMemoryParser (Int)	p27

Table 2. Header definitions

Fields	Type	
IDService	Int	h1
Subnet mask	String	h2
SizeMessage	Int	h3
NTimeRouting	Int	h4
LengthSOAPAction	Int	h5
TFMessageSent	Int	h6

Table 3. Definition of fields recovered by the parser

Fields	Type	variable
Probabilidad Oversize Payload	Real	x11
Attack Oversize Payload	Boolean	x12
Probabilidad Recursive Parsing	Real	x21
Attack Recursive Parsing	Boolean	x22
Probabilidad Buffer Overflow Attack	Real	x31
Attack Buffer Overflow Attack	Boolean	x32
Probabilidad XML Injection Attack	Real	x41
Attack Buffer Overflow Attack	Boolean	x42
Probabilidad Xpath Injection Attack	Real	x51
Attack Xpath Injection Attack	Boolean	x52
Probabilidad SQL Injection Attack	Real	x61
Attack SQL Injection Attack	Boolean	x62

Table 4. Fields stored as results

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:

$$c = (\{h_i / i = 1..6\} \cup \{p_j / j = 1..27\} \cup \{x_{lm} / l = 1.., m = 1..2\})$$

To store cases a hash table is used where the temporal execution cost associated to the different operations over the table is lineal, O(number of cases).

When the system receives a new request, the TB-CBR performs an analysis that can determine whether it is an attack, in that case it identifies the type of attack. To do it, in the Retrieve phase, the real-time agent recovers the cases that it will use to perform the classification. The time needed to recover the different cases to be used is clearly defined and temporal bounded. 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}, and it is defined by the following tuple: $c_{n+1} = (\{h_i / i = 1..6\})$. The new case does not initially contain information related to the parser since it would be necessary to analyze the content of the message:

$$c_{.h_1 h_2} = f_s(C) = \{c_{j.h_1 h_2} \in C / c_{j.h_1} = c_{n+1.h_1} \cap c_{j.h_2} = c_{n+1.h_2}\}$$

where $c_{j.h_1}$ 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.

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 the request. 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.

At the beginning of the Reuse phase, a number of different techniques 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.

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. 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. But 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 the design-to-time [23] planning, which assumes that multiple methods exist to complete various tasks and the problem consists of designing a solution that uses all of the resources to maximize the quality of the response within the available time. In order to determine which is the set of techniques that 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. The *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. To select the combination of techniques that produce the most optimal results are used multiple methods technique [13]. This technique selects the optimal combination given the amount of time available. 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 the analysis is not performed. The classifier agent in this case should 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 pc and are defined as follows: pc={ p1, p28, p29, p2, p3, p4, p5, p6, p16, p17, p26, p27}. The remaining entries vary according to the techniques used, which is specified for each one. Table 5 shows the information of the different techniques used for each of the attacks.

	Light Technique	Heavy Technique
Oversize Payload	Decission Tree	Neural Network
Recursive Parsing	Naïve Bayes	Red neuronal
Buffer Overflow Attack	Decision tree	Red neuronal
XML Injection Attack	SMO	Neural Network
Xpath Injection Attack		Neural Network
SQL Injection Attack		Neural Network

Table 5. Techniques associated with the attacks

The information used by the different techniques varies according to the type of attack. Table 6 details the different fields associated with each of the attacks for the Heavy techniques. The Light techniques only use the fields that refer to the request headers, specifically the following fields {h3, h4, h5, h6}.

Fields	Variable
Recursive Parsing	{h3, h4, h5, h6, pc}
Oversize Payload	{h3, h4, h5, h6, pc}
XML Injection Attack	{h3, h4, h5, h6, pc}
Buffer Overflow Attack	{h3, h4, h5, h6, pc}
SQL Injection Attack	{p8, p9, p10, p11, p12, p13, p14, p15} U {h3, h4, h5, h6, pc}
Xpath Injection Attack	{p19, p20, p21, p22, p23, p24} U {h3, h4, h5, h6, pc}

Table 6. Inputs associated with the different attack mechanisms

In addition to the techniques displayed in Table 5, there is a global neural network that contains each of the inputs listed in table 6 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.

Figure 1 shows the mixture of experts as it is carried out. It is possible to see how the input data gathered from the cases are distributed between the different attack classification techniques. Figure 1 shows only one part of the mixture. The complete description of the techniques associated with each attack is shown in table 5.

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 objective at which the attack was targeted (database, parser, etc.).

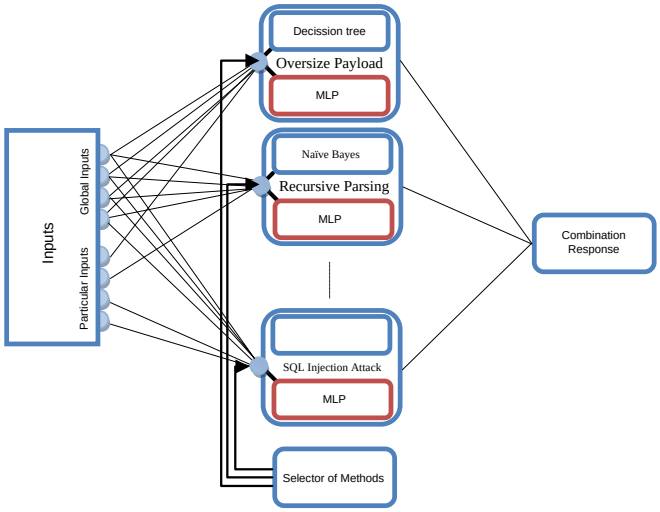

Figure 1. Multiple method inputs and outputs

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.

4 Validate Test: Detection of Attacks

In order to evaluate our prototype, we used a description of three services available from an application currently under development for testing purposes. Table 7 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 in 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 100Mbps Ethernet network connection. The PC used by the classifiers was an HP Pavilion Intel Core 2 Duo E7200 with 4GB 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.

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 the doctor via Internet	
IdDoctor	Int
PatientName	String
Date_Appointment	Date
Time_Appointment	Date
Descripción	String

Table 7. Description of the Web services used for the tests

Both the requesting agent and the real time classifier agent are executed over the *jART platform* [34], 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 the real time classifier agent must carry out. At the same time, both the agent and the platform work in a real-time operating system.

In order to analyze the final efficiency of the system, we forced the system 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 C1 combination (the fastest), and another slightly greater value that the agent needs to complete the analysis indicated by the combination C16. Table 8 shows the results obtained. Clearly, the unique algorithm created by a unique neural that can produce an estimate of attack under all the worst case input parameters is the estimate for the worst case, but the fastest estimate as well. The fact that it is the fastest is that the fastest combination C1 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 C16, which is the heaviest.

	Correct	Suspicious	Incorrect	Time
Unique algorithm	1457	34	9	0,563ms
C1	1463	14	23	0,611ms
C16	1484	12	4	1,722ms

Table 8. Number of responses that are correct, suspicious or incorrect, and the average execution time for different combinations

5 Conclusion

In addition to the techniques presented in this article, there are others that can perform similar classifications such as [35] and [36], that 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. To complete the analysis it could have been done in a less temporal restrictive manner, but there would be no guarantee of complying with all the temporal restrictions although similar classification results would be obtained.

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 regards 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, by the Spanish government under grants CONSOLIDER-INGENIO 2010 CSD2007-00022 and TIN2009-13839-C03, by GVA funds under PROMETEO 2008/051 project and the Professional Excellence Program 2006-2010 IFARHU-SENACYT-Panama.

References

[1] A. Nadalin, C. Kaler, R. Monzillo et al., *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, 2006.

[2] G. Della-Libera, M. Gudgin, P. Hallam-Baker et al., *Web services security policy language Version 1.0 (WS-SecurityPolicy)*, 2005.

[3] S. Anderson, J. Bohren, T. Boubez et al., *Web Services Trust Language (WS-Trust)*, 2004.

[4] S. Anderson, J. Bohren, T. Boubez et al., *Web Services Secure Conversation Language (WS-SecureConversation) Version 1.1*, 2004.

[5] N. Gruschka, M. Jensen, and N. Luttenberger, *A Stateful Web Service Firewall for BPEL*, In: IEEE International Conference on Web Services, pp. 142-149. 2007.

[6] C. L. Schuba, I. V. Krsul, M. G. Kuhn et al., *Analysis of a Denial of Service Attack on TCP*, In: Proceedings of the IEEE Symposium on Security and Privacy (SP'97),p. 208, 1997.

[7] E. G. Im, and Y. H. Song, An Adaptive Approach to Handle DoS Attack for Web Services, In:S. B. Heidelberg, ed. Intelligence and Security Informatics Lecture Notes in Computer Science, pp. 634-635, 2005.

[8] K. Zhao, K. Yang, M. Zhang et al., *Denial of Service Attack Simulation Based-on CASL*, In: IEEE International Workshop on Anti-counterfeiting, Security, Identification, pp. 266-269, 2007.

[9] E. Pulier, and H. Taylor. *Understanding Enterprise SOA*, Greenwich, CT, USA: Manning Publications Co., 2005.

[10] E. Cerami. *Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSD*, First Edition O'Reilly & Associates, 2002.

[11] M. Gudgin, M. Hadley, N. Mendelsohn et al., *W3C Recommendation: SOAP Version 1.2 Part 2: Adjuncts* (Second Edition), 2007.

[12] R. A. Jacobs, M. I. Jordan, S. J. Nowlan et al., *Adaptive mixtures of local experts*, Neural Computing, vol. 3, no. 1, 79-87, 1991.

[13] A. J. Garvey, and V. R. Lesser, *Design-to-time real-time scheduling*, Systems, Man and Cybernetics, IEEE Transactions on, vol. 23, no. 6, 1491-1502, 1993.

[14] A. Subasi, *EEG signal classification using wavelet feature extraction and a mixture of expert model*, Expert Systems with Applications, vol. 32, no. 4, 1084-1093, 2007.

[15] E. Moradian, and A. Håkansson, *Possible attacks on XML Web Services*, International Journal of Computer Science and Network Security (IJCSNS), vol. 6, no. 1B, 154-170, 2006.

[16] R. Bebawy, H. Sabry, S. El-Kassas et al., Nedgty: *Web Services Firewall*, In: IEEE International Conference on Web Services, pp. 597-601, 2005.

[17] J. Wang, *Defending Against Denial of Web Services Using Sessions*, In: IEEE/IST Workshop on: Monitoring, Attacking Detection and Mitigation, 2006.

[18] Y.-S. Loh, W.-C. Yau, C.-T. Wong et al., *Design and Implementation of an XML Firewall*, In: International Conference on Computational Intelligence and Security, pp. 1147-1150, 2006.

[19] S. Padmanabhuni, V. Singh, K. M. S. Kumar et al., *Preventing Service Oriented Denial of Service (PreSODoS): A Proposed Approach*, In: IEEE Int. Conference on Web Services, pp. 577-584, 2006.

[20] C. G. Yee, W. H. Shin, and G. S. V. R. K. Rao, *An Adaptive Intrusion Detection and Prevention (ID/IP) Framework for Web Services*, In: Int. Conference on Convergence Information Technology, pp. 528-534, 2007.

[21] M. Srivatsa, A. Iyengar, J. Yin et al., *Mitigating application-level denial of service attacks on Web servers: A client-transparent approach*, ACM Trans. Web, vol. 2, no. 3, 1-49. 2008.

[22] X. Ye, *Countering DDoS and XDoS Attacks against Web Services*, In: IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, pp. 346-352, 2008.

[23] A. Chonka, W. Zhou, and Y. Xiang, *Defending Grid Web Services from XDoS Attacks by SOTA*, In: EEE International Conference on Pervasive Computing and Communications , pp. 1-6, 2009.

[24] V. Julian, and V. Botti, Developing real-time multi-agent systems, Integrated Computer-Aided Engineering, vol. 11, no. 2, pp. 135-149, 2004.

[25] J. A. Stankovic, *Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems*, IEEE Computer, vol. 21, no. 10, pp. 10-19, 1988.

[26] A. Garvey, and V. Lesser, *A survey of research in deliberative real-time artificial intelligence*, Real-Time Systems, vol. 6, no. 3, 317-347, 1994.

[27] V. J. Botti, C. Carrascosa, J. Vicente et al., Modelling Agents in Hard Real-Time Environments, In: 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, pp. 63—76, 1999.

[28] A. Martens, and A. M. Uhrmacher, *Adaptative tutoring processes and mental plans*, In Proceedings of Intelligent Tutoring Systems—ITS, pp. 71-80, 2002.

[29] A. Aamodt, and E. Plaza, *Case-based reasoning: foundational issues, methodological variations, and system approaches*, AI Communications, vol. 7, no. 1,39-59, 1994.

[30] T. Dean, and M. S. Boddy, *An Analysis of Time-Dependent Planning*, In: 7th National Conference on Artificial Intelligence, pp. 49-54, 1988.

[31] D. W. Patterson, M. Galushka and N. Rooney, *Characterization of a novel indexing technique for case-based reasoning*, Artificial Intelligence Review 23, pp. 359-393, 2005.

[32] J. R. Quinlin, *C4.5 Programs for Machine Learning*, Morgan Kauffman, 1993.

[33] S. Wess, K. D. Althoff and M. Richter, *Using k-d trees to improve the retrieval step in case-based reasoning*, in: European Workshop, Topics in Case-based Reasoning, pp. 67-81, 1993.

[34] M. Navarro, V. Julian, J. Soler et al., *jART: A Real-Time Multi-Agent Platform with RT-Java*, In: 3th International Workshop on Practical Applications of Agents and Multi-Agent Systems, pp. 73-82, 2004.

[35] H. R. Bittencourt, and 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), pp. 3751-3753, 2003.

[36] W. W. Cohen, *Fast Effective Rule Induction*, In: In Proceedings of the Twelfth International Conference on Machine Learning, pp. 115-123, 1995.

[37] M. Navarro, S. Heras and V. Julián, *Guidelines to Apply CBR in Real-Time Multi-Agent Systems*, In: Journal of Physical Agents 3(3), pp. 39-43, 2009.