A Hybrid Agent-based Classification Mechanism to Detect Denial of Service Attacks

Cristian I. Pinzón, Juan F. De Paz, Sara Rodríguez, Javier Bajo and Juan M. Corchado

***Abstract*—This paper presents the core component of a solution based on agent technology specifically adapted for the classification of SOAP messages. The messages can carry out attacks that target the applications providing Web Services. One of the most common attacks requiring novel solutions is the denial of service attack (DoS), caused for the modifications introduced in the XML of the SOAP messages. The specifications of existing security standards do not focus on this type of attack. This article presents an advanced mechanism of classification designed in two phases incorporated within a CBR-BDI Agent type. This mechanism classifies the incoming SOAP message and blocks the malicious SOAP messages. Its main feature involves the use of decision trees, fuzzy logic rules and neural networks for filtering attacks. These techniques provide a mechanism of classification with the self-adaption ability to the changes that occur in the patterns of attack. A prototype was developed and the results obtained are presented in this study.**

***Index Terms*—multi-agent systems, case-based reasoning, DoS Attack, SOAP message, XML security.**

I. Introduction

THE communication based on Service Oriented Architecture Web (SOA) is carried out by XML-based messages, called SOAP messages. This message exchange process is one of the key elements required in SOA environments for system integration [1]. The SOAP message payload often consists of sensitive information, which is sent through insecure channels such as HTTP connections. If a malicious user playing the role of a middleman intercepts a message between sender and recipient, it can result in a series of malicious tasks carried out over the captured message.

DoS attacks are due to the fact that XML messages must be parsed in the server, which opens the possibility of an attack if the messages themselves are not well structured or if they include some type of malicious code. Resources available in the server (memory and CPU cycles) of the provider can be drastically reduced or exhausted while a malicious SOAP message is being parsed. A DoS attack is successfully carried out when it manages to severely compromise legitimate user access to services and resources.

A number of technologies and solutions have been proposed for addressing the secure exchange of SOAP message such as WS-Security [2], WS-SecurityPolicy [3], WS-Trust [4], WS-SecureConversation [5] etc. All these standards focus on the aspects of message integrity and confidentiality and user authentication and authorization. None of the WS-Security Standards provide full security, leaving gaps that can be exploited by any malicious user.

Then, it is necessary to investigate in novel methods to protect the servers from denial of services attacks (DoS), which cause malicious or altered Web Services, and affect the availability of the Web Services [6]. This paper presents the core component of a strong solution based on a multi-agent architecture for tackling the security issue of the Web Service. This core is embedded in a CBR-BDI [7] deliberative agent based on the BDI (Belief, Desire, Intention) [8] model specifically adapted for preventing many attacks over web services. Our study applies a solution in two phases that include novel case-based reasoning (CBR) [9] classification mechanisms. The first phase incorporates decision tree and fuzzy logic rules [10] while the second phase incorporates neural networks capable of making short term predictions [11]. The idea of a CBR mechanism is to exploit the experience gained from similar problems in the past and to adapt a successful solution to the current problem. The CBR engine initiates what is known as the CBR cycle, which is comprised of 4 phases. The CBR-BDI agent explained in this work uses the CBR concept to gain autonomy and improve its problem-solving capabilities. The approach presented in this paper is entirely new and offers a different way to confront the security problem in SOA environments.

The rest of the paper is structured as follows: section 2 presents the problem that has prompted most of this research. Section 3 focuses on the structure of the classifiers agents which facilitates classification of SOAP message, and section 4 provides a detailed explanation of the classification model integrated within the type of classifier agent. Finally, section 5 presents the conclusions obtained by the research.

II. Web Service Security Problem Description

A web service is a software module designed to support interaction between heterogeneous groups within a network. In order to obtain interoperability between platforms, communication between web servers is carried out via an exchange of messages. These messages, referred to as SOAP messages, are based on standard XML (eXtensible Markup Language) and are primarily exchanged using HTTP (Hyper Text Transfer Protocol) [12].

One of the most frequent techniques of a DoS attack is to exhaust available resources (memory, CPU cycles, and

Cristian I. Pinzon is with The Technological University of Panama.
E-mail: cristian.pinzon@utp.ac.pa
Juan F. De Paz is with University of Salamanca.
E-mail: fcofds@usal.es
Sara Rodríguez is with University of Salamanca.
E-mail: srg@usal.es
Javier Bajo is with University of Salamanca.
E-mail: jbajope@usal.es
Juan M. Corchado is with University of Salamanca.
E-mail: corchado@usal.es

bandwidth) on the host server. The probability of a DoS attack increases with applications providing web services because of their intrinsic use of the XML standard. The server uses a parser, such as DOM, Xerces, etc. to syntactically analyze all incoming XML formatted SOAP messages. When the server draws too much of its available resources to parse SOAP messages that are either poorly written or include a malicious code, it risks becoming completely blocked. Attacks usually occur when the SOAP message either comes from a malicious user or is intercepted during its transmission by a malicious node that introduces different kinds of attacks.

Security is one of the greatest concerns within web service implementations. The following list contains descriptions of different types of attacks, compiled from those noted in [13], [14], [15].

- Oversize Payload: When it is executed, it reduces or eliminates the availability of a web service while the CPU, memory or bandwidth are being tied up by a massive message dispatch with a large payload.
- Coercive Parsing: Just like a message written with XML, an XML parser can analyze a complex format and lead to a denial of service attack because the memory and processing resources are being used up.
- Injection XML: This is based on the ability to modify the structure of an XML document when an unfiltered user entry goes directly to the XML stream or the message is captured and modified during its transmission.
- Parameter Tampering: A malicious user employs web service entries to manually or automatically (dictionaries attack) execute different types of tests and produce an unexpected response from the server.
- SOAP header attack: Some SOAP message headers are overwritten while they are passing through different nodes before arriving at their destination. It is possible to modify certain fields with malicious code.
- Replay Attack: Sent messages are completely valid, but they are sent en masse over a small time frame in order to overload the web service.

All web service security standards focus on strategies independent from DoS attacks [6]. In the following, we will revise those works that focus on denial of web service attacks and will compare to our approach.

A "XML Firewall"is proposed by [13]. The architecture of the XML Firewall is divided into three modules, namely 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 the authenticity of the contents. If the contents of the messages do not conform to the policies that have been set, the messages will be dropped by the firewall. Three successfully implemented filtering policies, namely message size filtering, syntax parsing, and XML schema validation have been tested with valid and invalid SOAP messages. Gruschka and Luttenberger [6] 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 and these are usually defined by an XML Schema, written in the 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 concrete type of attack inside Web Services. An adaptive framework for the prevention and detection of intrusions was presented in [14]. Based on a hybrid focus that combines agents, data mining and fuzzy logic, it is supposed to filter attacks that are either already known or new. Agents that act as sensors are used to detect violations to the normal profile using the data mining technique 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. An approach to countering DDoS and XDoS attacks against web services is presented by [16]. The system carries out request message authentication and validation before the requests are processed by the Web Services providers. The scheme has two modes: the 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 [17] 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 message. SOTA is a traceback system that is constructed on the basis of Web Services and is able to traceback to the source of the malicious message. Once an attack has been discovered and the attacker's identity known, XDetector can filter out these attack messages.

Our approach outperforms the existing models with respect to:

- Learning and Adaptive ability: These features are the most important of our approach. Our approach includes one type of intelligent agents that was designed to learn and adapt to changes in attack patterns and new attacks.
- Tolerance to Failure: Our approach has a design that can facilitate error recovery through the instantiation of new agents.
- Scalability: Our approach is capable of growing (by means of the instantiation of new agents) according to the needs of its environment.

Our approach presents novel characteristics that have not heretofore been considered in previous approaches. The next section presents the architecture in greater detail. The following sections detail the internal model of the CBR-BDI agent, as well as the classification process for SOAP message for identifying malicious messages.

III. Classifiers Agents Internal Structure

Agents are characterized by their autonomy; which gives them the ability to work independently and in real-time

environments [18]. Because of this and their other capacities, agents are being integrated into security approaches such as Intrusion Detection Systems (IDS) [19]. However, the use of agents in these systems focuses on the retrieval of information in distributed environments, which only takes advantage of their mobility capacity.

The classification agent presented in this study interacts with other agents within the architecture. These agents carry out tasks related to capturing messages, syntactic analysis, administration, and user interaction. As opposed to the tasks for these agents, the classification agent executes a classification of SOAP messages in two phases that we will subsequently define in greater detail.

In our research, the agents are based on a BDI model in which beliefs are used as cognitive aptitudes, desires as motivational aptitudes, and intentions as deliberative aptitudes in the agents [8]. However, in order to focus on the problem of the SOAP message attack, it was necessary to provide the agents with a greater capacity for learning and adaption, as well as a greater level of autonomy than a pure BDI model currently possesses. This is possible by providing the classifier agents with a CBR mechanism [9], which allows them to "reason" on their own and adapt to changes in the patterns of attacks.

Case-based Reasoning (CBR) is a type of reasoning based on the use of past experiences [9]. The purpose of case-based reasoning systems is to solve new problems by adapting solutions that have been used to solve similar problems in the past. The fundamental concept when working with case-based reasoning is the concept of case. A case can be defined as a past experience, and is composed of three elements:

- A problem description which describes the initial problem.
- A solution which provides the sequence of actions carried out in order to solve the problem.
- The final state which describes the state achieved after the solution was applied.

A case-based reasoning system manages cases (past experiences) to solve new problems. The way in which cases are managed is known as the case-based reasoning cycle. These systems execute the CBR cycle which consists of four sequential steps: retrieve, reuse, revise and retain [9]. The method proposed in [20] facilitates the incorporation of case-based reasoning systems as a deliberative mechanism within BDI agents, allowing them to learn and adapt themselves, lending them a greater level of autonomy than what is normally found in a typical BDI architecture [21]. Accordingly, our Classifiers agents can reason autonomously and therefore adapt themselves to changes in the attack patterns. The case-based reasoning system is completely integrated within the Classifiers agents. These agents incorporate a "formalism" that is easy to implement, in which the reasoning process is based on the concept of intention. Intentions can be seen as cases, which have to be retrieved, reused, revised and retained. A direct relationship between case-based reasoning systems and BDI agents can also be established if the problems are defined in the form of states and actions.

Case: <Problem, Solution, Result>	**BDI agent**
Problem: initial_state	Belief: state
Solution: sequence of <action,[intermediate_state]>	Intention: sequence of <action>
Result: final_state	Desire: set of <final_state>

Our Classifiers agents implement cases as beliefs, intentions and desires which lead to the resolution of the problem. As described in [22], [23], each state of a CBR-BDI agent is considered as a belief, including the objective to be reached. The intentions are plans of actions that the agent has to carry out in order to achieve its objectives, which makes each intention an ordered set of actions. Each change from state to state is made after carrying out an action (the agent remembers the action carried out in the past, when it was in a specified state, and the subsequent result). A desire will be any of the final states reached in the past (if the agent has to deal with a situation that is similar to one from the past, it will try to achieve a result similar to the one previously obtained). The Classifiers agents used in our solution, use these concepts to define a case structure for DoS attacks in SOAP messages.

As previously mentioned, the classifier CBR-BDI agent is the core of the multi-agent architecture and is geared towards classifying SOAP messages for detecting attacks on web services. Figure 1 shows the classifier CBR-BDI agents in each phase of the mechanism of classification. These CBR-BDI classifier agents will be explained in detail in next section.

IV. Mechanism for the Classification of SOAP Message Attack

The CBR-BDI classifier agent presented in section 3 incorporates a case-based reasoning mechanism that allows it to classify SOAP messages. The mechanism incorporated into the agent approaches the idea of classification from the perspective of anomaly-based detection. In the specific case of SOAP messages, it manages a case memory for each service offered by the Web Service environment, which permits it to handle each incoming message based on the particular characteristics of each web service available. Each new SOAP message sent to the architecture is classified as a new case study object. Focusing on the problem that is of interest to us, we will represent a typical SOAP message which consists of a type of wrapping that contains an optional heading and a mandatory body of text with a useful message load, as depicted in figure 2.

Based on the structure of the SOAP messages and the transport protocol used, we can obtain a series of descriptive fields to consider. Based on this information, we can present a two-part strategy for executing the classification process:

A. First Phase of the mechanism of Classification

The main goal of this initial phase is to carry out an effective classification, but without requiring an excessive amount of resources and time. As a CBR strategy is used, it is necessary to define the case structure used by the classifiers CBR-BDI agents. The fields of the case are obtained from the headers of the packages of the HTTP\TCP-IP transport protocol. Table I shows the fields taken into consideration to describe the problem.

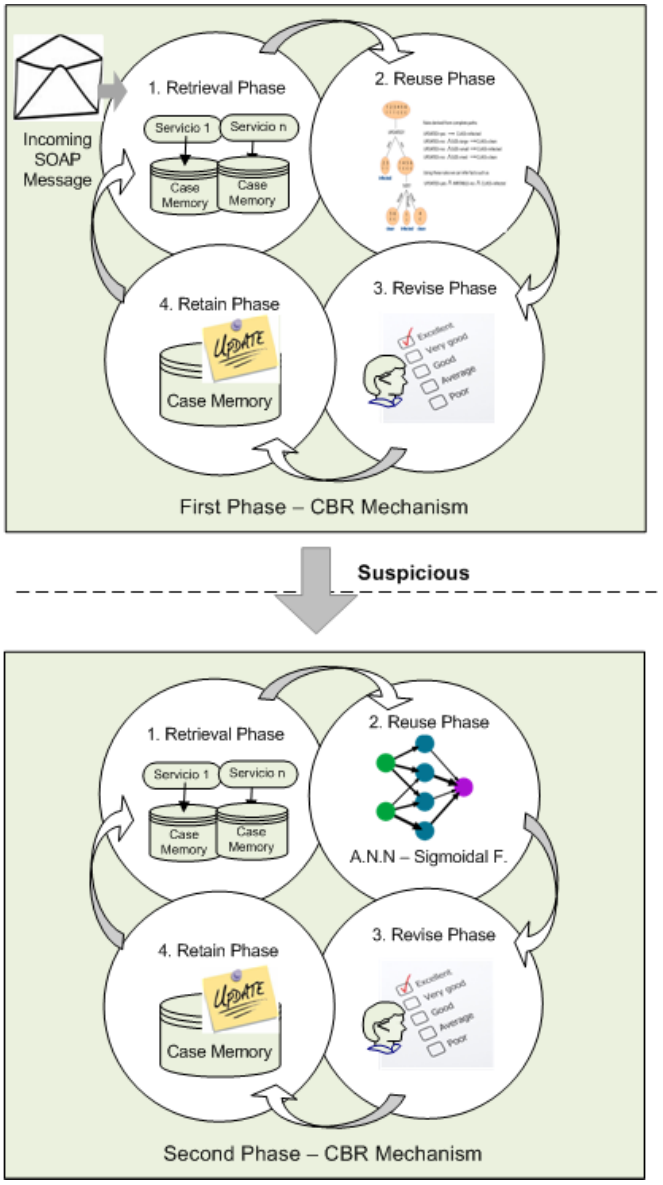

Fig. 1. Design of the mechanism of classification - Classifier CBR-BDI agents

Fig. 2. Content and Structure of a SOAP Message

TABLE I
CASE DESCRIPTION - CBR - FIRST PHASE

Fields	Type	Variable
IDService	Int	i
SubnetMask	String	m
SizeMessage	Int	s
NTimeRouting	Int	n
LengthSOAPAction	Int	l
TFMessageSent	Int	w

As can be seen in Table I, the description of a case is given by the tuple $c = (i, m, s, n, l, w, R/C_{.im}, x^p, x^r)$, where i represents the service identifier, m the subnet mask, s the message length, n the number of seconds for the travel of the message, l the length of the header SoapAction, w the elapsed time from the arrival of the last n messages, $R/C_{.im}$ is the solution provided by the decision tree associated to the service and to the subnet mask, x^p represents the class predicted by the CBR strategy $x^p \in X = \{a, g, u\}$, where a,g,u represent the values attack, good and undefined, x^r is the real class $x^r \in X = \{a, g, u\}$.

The CBR strategy is integrated into a BDI agent, obtaining a CBR-BDI agent. The integration of the CBR system and the BDI agent is defined as follows: beliefs - problem description and rules; intentions - set of accions and rules that represent the state transitions required to achieve the final state; desires - $X = \{a, g, u\}$. The initial state is defined by means of the set of beliefs that store the values for the subnet mask and the service web identifier, $(i, m, \phi, \phi, \phi, \phi)$. The intermediate states describe the decision process executed, taking into account the application of rules over the set of rules.

The cases memory contains a set of cases $C = \{c\}$ and is fragmented for each of the web services available in the server. This structure facilitates the depuration and analysis of the services in an independent manner. Separately to the cases memory, the agent incorporates a rules memory, constructed as a set of inductive rules defined as $R = \{r_1, \ldots, r_1\}$ with $r_i = (l_1 \wedge \ldots \wedge l_m) \rightarrow x_j$ where $l_s = (d_{ts}, o_s, \Re)/d_{ts} \in (i, m, s, n, l, w, x^p, x^r)$, $o_s \in O$, with $O = \{=, \neq, \succ, \prec, \leq, \geq\}$, $x_j \in X$. The rules memory is also fragmented for each of the services and for each of the subnet mask, in a way that $R/C_{.im}$ represents the rules associated to those cases belonging to the service i and the subnet mask m. For notation considerations, to identify a property of a case, we use the case, a point and the property. For example, $c_{j.m}$ represents the property m (subnet mask) of the case j.

When the agent receives a request to classify a new case c_{n+1}, a new execution of a CBR cycle is carried out. The following paragraphs describe the stages of a CBR cycle executed in the first phase classification.

- Retrieve: During this stage, those cases associated to the requested web service and the corresponding rules memory are retrieved. The storage and recovery of rules from the rules memory facilitates a notably reduction of the process time for the classification. The retrieve strategy is carried out as follows:
  - If there is not tree associated to the service and the subnet mask, then it is necessary to recover the cases for the service and the subnet mask:

$$C_{.im} = f_s(C) = \{c_{j.im} \in C / c_{j.i} = c_{n+1.i}, c_{j.m} = = c_{n+1.m}\} \quad (1)$$

    Where $C_{j.i}$ represents the case j and i the service identifier.
  - The rules memory associated with the set of cases $R/C_{.im}$ is retrieved
- Reuse: Knowledge extraction is especially important when complex algorithms that use hard computing techniques and that generate models in an automatic way are used. Human experts are much confident when they know exactly why or at least how a solution to a problem has been calculated. Classification And Regression Tree (CART) is a nonparametric statistical method for extraction of knowledge in classifications. The extracted information is represented in a binary decision tree, which allows individuals to be classified from the root node. Keeping the kind of dependent variable in mind, CART can be separated into two types: classification tree, if the dependent variable is categorical; and regression tree in the case of a continuous dependent variable.
  The reuse stage is only executed if not decision tree $R/C_{.im}$ associated to the cases $c_{.im}$ is available, and in order to do so, the rules are generated using the CART algorithm. $R/C_{.im} = CART(c_{.im})$ where $R/C_{.im}$ is the rules memory associated to the service identifier and to the subnet mask. The CART algorithm has been modified in order to have an automatic discretization of the values to a set of categories. The modification includes a first step to normalize the variable into the interval $[0, 1]$ and then, the values are discretized into one of the following categories depending on the closest value {very low=0.1, low=0.3, medium=0.5, high=, 0.7 y very high=0.9}. This way, the generation of rules using the CART algorithm is more efficient than working with a greater level of categories. The discretization is only carried out for the variables s, n, l, w.
- Revise: Once the set of rules has been retrieved, the classification for the case $c_{n+1.im}$ is obtained using the set of rules that previously classified the elements of the same type $c_{n+1.x^p = R/C_{.im}(c_{n+1})}$. If $r_i \in R/C_{.im}$ then, it is the rule that classifies c_{n+1}. The new case is classified as follows:
  - If $m_i \succ \mu_1 \| \#\{c_j \in C_{r_j} / c_{j.x^p} = u\} \succ \mu_2$ then, it is necessary to execute the CBR of the second phase. Where $C_{r_i} \subseteq C$ is the set of cases classified for r_i and m_i represents the percentage of misclassified cases of C_{r_j} using the rule r_j. # represents the number of elements of the set. The general idea is to verify if the error rate of the rule exceeds a certain threshold, and then, verify that the number of cases belonging to the set of elements classified using the rule not exceeds a certain threshold defined as a function of the total number of elements in C_{r_j}.
  - Else if $\#\{c_j \in C_{r_j} / c_{j.x^p} = g\} / \#C_{r_j} \succ \alpha_g$ then the case is classified as good and the revision finishes.
  - Else if $\#\{c_j \in C_{r_j} / c_{j.x^p} = u\} / \#C_{r_j} \succ \alpha_s$ the case is classified as suspicious and the second phase classification mechanism is executed.
  - Else if $\#\{c_j \in C_{r_j} / c_{j.x^p} = a\} / \#C_{r_j} \succ \alpha_a$ the case is classified as attack and the revision finishes.
- Retain: If the set of rules was generated because it didn't previously exist, then $R/C_{.im}$ is stored in the rules memory if the classification obtained was good. If the classification was erroneous and the misclassification was detected by an expert or if the second phase was invoked, then it is necessary to regenerate the decision tree: $R/C_{.im} = CART(c_{.im} \cup c_{n+1})$.

B. Second Phase of the mechanism of Classification

The fields are extracted from the SOAP message and provide the case description for second phase of the mechanism of classification. Table II presents the fields used in describing the problem for the CBR in this layer.

TABLE II
CASE DESCRIPTION - CBR- SECOND PHASE

Fields	Type	Variable
IDService	Int	i
SubnetMask	String	m
SizeMessage	Int	s
NTimeRouting	Int	n
LengthSOAPAction	Int	l
MustUnderstandTrue	Boolean	u
NumberHeaderBlock	int	h
NElementsBody	int	b
NestingDepthElements	int	d
NXMLTagRepeated	int	t
NLeafNodesBody	int	f
NAttributesDeclared	int	a
CPUTimeParsing	int	c
SizeKbMemoryParser	int	k

Applying the nomenclature shown in the table above, each case description is given by the following tuple:

$$c = (i, m, s, n, l, u, h, b, d, t, f, a, c, k, P/C_{.im}, x^p, x^r) \quad (2)$$

For each incoming message received by the agent and requiring classification, we will consider both the class that the agent predicts and the class to which the message actually belongs. x^p represents the class predicted by the classifier agent belonging to the group. $x^p \in X = \{a, g, u\}$; g and u represent attack, good and undefined, respectively; and x^r is the class to which the attack actually belongs, $x^r \in X = \{a, g, u\}$, $P/C_{.im}$ is the solution provided by the neural network Multilayer perceptron (MLP)associated to service i and subnet mask m.

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 neural network. Each P_i is a reasoning memory related to a group of cases dependent of the service and subnet mask of the client. The Multilayer Perceptron (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 [24]. The 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 the 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:

$$f(x) = \frac{1}{1 + e^{-ax}} \quad (3)$$

The CBR mechanism executes the following phases:

- Retrieve: the cases that are most similar to the current problem, considering both the type of Web service to which the message belongs and the subnet mask that contains the message.

$$C_{.im} = f_s(C) = \{c_j \in C / c_{j.i} = c_{n+1.i}, \; c_{j.m} = c_{n+1.m}\} \quad (4)$$

Once the similar cases have been recovered, the neural network MLP $P/C_{.im}$ associated to service i and subnet mask m is then recovered.

- Reuse: The classification of the message is begun in this phase, based on the subnet mask and the recovered cases. It is only necessary to retrain the neural network when it does not have previous training. The entries for the neural network correspond to the case elements $s, n, l, u, h, b, d, t, f, a, c, k$. 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. The training for the network is carried out by the error Backpropagation Algorithm [25]. The weights and biases for the neurons at the exit layer are updated by following equations:

$$w^p_{kj}(t+1) = w^p_{kj}(t) + \eta(d^p_k - y^p_k)(1 - y^p_k)y^p_k y^p_j + \\ +\mu(w^p_{kj})(t) - w^p_{kj}(t-1)) \quad (5)$$

$$\theta^p_k(t+1) = \theta^p_k(t) + \eta(d^p_k - y^p_k)(1 - y^p_k)y^p_k + \mu(\theta^p_k(t) - \\ -\theta^p_k(t-1)) \quad (6)$$

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

$$w^p_{ji}(t+1) = w^p_{ji}(t) + \eta(1 - y^p_j)y^p_j(\textstyle\sum_{k=1}^{M}(d^p_k - y^p_k) \\ (1 - y^p_k)y^p_k w_{kj})x^p_i + \mu(w^p_{ji})(t-1)) \quad (7)$$

$$\theta^p_j(t+1) = \theta^p_j(t) + \eta(1 - y^p_j)y^p_j(\textstyle\sum_{k=1}^{M}(d^p_k - y^p_k) \\ (1 - y^p_k)y^p_k w_{kj}) + \mu(\theta^p_j(t) - \theta^p_j(t-1)) \quad (8)$$

where w^p_{kj} 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^p_k represents the desired value, y^p_k the value obtained for neuron k from the exit layer, y^p_j the value obtained for neuron j from the intermediate layer, η the learning rate and μ the momentum. θ^p_k represents the bia value k from the exit layer. The variables for the intermediate layer are defined analogously, keeping in mind that i represents the neuron from the entrance level, j is the neuron from the intermediate level, M is the number of neurons from the exit 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 beginning with the cases related to the service and subnet mask, as shown in equation 9.

$$P_r = MLP^t(c_{.im}) \tag{9}$$

- Revise: This phase reviews the classification performed in the previous phase. The value obtained by exiting the network $y = P_r^e(c_{n+1})$ may yield the following situations:
  - If $y \succ \mu_1$ then it is considered an attack.
  - Otherwise, if $y \prec \mu_2$, then the message is considered a non-attack or legal.
  - Otherwise, the message is marked as suspicious and is filtered for subsequent revision by a human expert. To facilitate the revision, an analysis of the neural network sensibility is shown so that the relevance of the entrances can be determined with respect to the predicted value
- If the result of the classification is suspicious or if the administrator identifies the classification as erroneous, then the network repeats the training by incorporating a new case and following the BackPropagation training algorithm.

$$P_r = MLP^t(c_{.im} \cup c_{n+1}) \tag{10}$$

The next section presents the conclusions and results obtained of a developed prototype of our mechanism of classification.

V. Conclusion

This research has presented the nucleus of a novel solution that focuses on the protection of web services. The focus incorporates case-based reasoning methods, decision trees, fuzzy logic rules, neural networks, and intelligent agent technology that allows us to approach the problem of web security from a perspective based on learning, adaptability and flexibility.

The solution was designed to be carried out in two phases. In the first phase, a CBR mechanism incorporates decision trees; fuzzy logic rules generate a preliminary robust solution regarding the condition of the message, without sacrificing application performance. If the obtained solution is classified as suspicious, we then proceed to the second phase of the process. This phase does involve a more complex process, with a greater need for resources, and where a second CBR mechanism embeds within a neural network to generate a final result.

A prototype of our proposed solution was based on a classification mechanism and developed in order to evaluate its effectiveness. The tests of the simulation were carried out within a small web application developed with Java Server Page and hosted in a Apache Tomcat 6.0.18 Server by using as web service engine, Apache Axis2 1.4.1. The tests were organized within 6 blocks with a specific number of requests (50, 100, 150, 200, 250 and 300) that allowed evaluating the effectiveness of the classifier agent in accordance with the gained experience. Within the blocks were included legal and illegal requests. Figure 3 shows the results obtained.

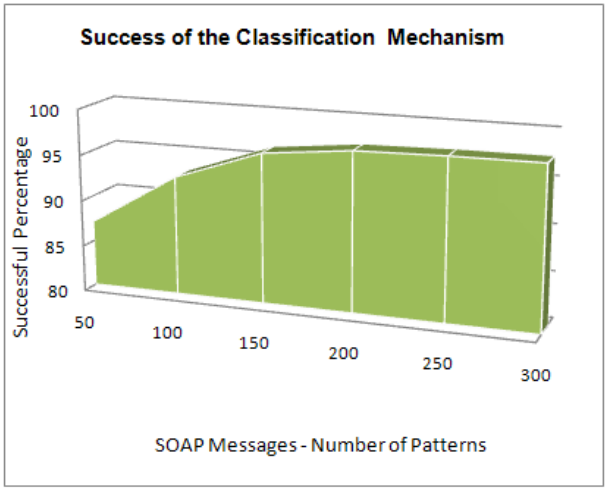

Fig. 3. Success of the Classification Mechanism

Figure 3 shows the percentage of prediction with regards to the number of patterns (SOAP messages) for the classification mechanism. It is clear that as the number of patterns increases, the success rate of prediction also increases in terms of percentage. This is influenced by the fact that we are working with CBR systems, which depend on a larger amount of data stored in the memory of cases.

The proposed solution will continue in the investigation and development for its application in various environments where its performance can be evaluated and real results obtained.

Acknowledgment

This development has been partially supported by the Spanish Ministry of Science project TIN2006-14630-C03-03 and The Professional Excellence Program 2006-2010 IFARHU-SENACYT-Panama

References

[1] M. A. Rahaman, A. Schaad, and M. Rits, "Towards secure soap message exchange in a soa," in *SWS '06: Proceedings of the 3rd ACM workshop on Secure web services*. New York, NY, USA: ACM, 2006, pp. 77–84.

[2] OASIS, "Web services security: Soap message security 1.1 (ws-security 2004)."

[3] G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, and C. Kaler, "Web services security policy language version 1.0 (ws-securitypolicy)," 2005.

[4] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della, and B. Dixon, "Web services trust language (ws-trust)," 2004.

[5] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, and B. Dixon, "Web services secure conversation language (ws-secureconversation) version 1.1." 2004.

[6] N. Gruschka and N. Luttenberger, "Protecting web services from dos attacks by soap message validation," in *SEC*, 2006, pp. 171–182.

[7] R. Laza, R. Pavn, and J. M. Corchado, "A reasoning model for CBR_BDI agents using an adaptable fuzzy inference system," in *10th Conference of the Spanish Association for Artificial Intelligence*, ser. Lecture Notes in Computer Science, R. Conejo, M. Urretavizcaya, and J. L. P. de la Cruz, Eds., vol. 3040. Springer, 2003, pp. 96–106.

[8] A. S. Rao and M. P. Georgeff, "Modeling rational agents within a BDI-architecture," in *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, pp. 473–484. [Online]. Available:

[9] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations, and system approaches," *AI Commun.*, vol. 7, no. 1, pp. 39–59, March 1994.

[10] H. Bittencourt and R. Clarke, "Use of classification and regression trees (cart) to classify remotely-sensed digital images," in *Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*, vol. 6, July 2003, pp. 3751–3753 vol.6.

[11] J. Shun and H. A. Malki, "Network intrusion detection system using neural networks," *International Conference on Natural Computation*, vol. 5, pp. 242–246, 2008.

[12] J. Snell, D. Tidwell, and P. Kulchenko, *Programming Web Services with SOAP*. O'Reilly, 2001.

[13] Y.-S. Loh, W.-C. Yau, C.-T. Wong, and W.-C. Ho, "Design and implementation of an xml firewall," *International Conference on Computational Intelligence and Security*, vol. 2, pp. 1147–1150, Nov. 2006.

[14] 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 *International Conference on Convergence Information Technology (ICCIT '07)*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 528–534.

[15] M. Jensen, N. Gruschka, R. Herkenhoner, and N. Luttenberger, "Soa and web services: New technologies, new standards - new attacks," *Fifth European Conference on Web Services*, pp. 35–44, Nov. 2007.

[16] X. Ye, "Countering ddos and xdos attacks against web services," in *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, vol. 1, 2008, pp. 346–352.

[17] A. Chonka, W. Zhou, and Y. Xiang, "Defending grid web services from xdos attacks by sota," in *IEEE International Conference on Pervasive Computing and Communications*, 2009, pp. 1–6.

[18] C. Carrascosa, J. Bajo, V. Julian, J. M. Corchado, and V. Botti, "Hybrid multi-agent architecture as a real-time problem-solving model," *Expert Syst. Appl.*, vol. 34, no. 1, pp. 2–17, 2008.

[19] A. Abraham, R. Jain, J. Thomas, and S. Y. Han, "D-scids: distributed soft computing intrusion detection system," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 81–98, 2007.

[20] J. M. Corchado and R. Laza, "Constructing deliberative agents with case-based reasoning technology," *International Journal of Intelligent Systems*, vol. 18, pp. 1227–1241, 2003.

[21] M. E. Bratman, D. J. Israel, and M. E. Pollack, "Plans and resource-bounded practical reasoning," in *Computational Intelligence*, vol. 4, 1988, pp. 349–355.

[22] J. Bajo, J. F. D. Paz, D. I. Tapia, and J. M. Corchado, "Distributed prediction of carbon dioxide exchange using cbr-bdi agents," *International Journal of Computer Science*, pp. 16–25, 2007.

[23] J. M. Corchado, M. Glez-Bedia, Y. D. Paz, J. Bajo, and J. F. D. Paz, "Replanning mechanism for deliberative agents in dynamic changing environments," *Computational Intelligence*, vol. 24, pp. 77–107, 2008.

[24] M. Gallagher and T. Downs, "Visualization of learning in multilayer perceptron networks using principal component analysis," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 33, no. 1, pp. 28–34, Feb 2003.

[25] Y. LeCun, L. Bottou, G. Orr, and K. Muller, "Efficient backprop," in *Neural Networks: Tricks of the trade*, G. Orr and M. K., Eds. Springer, 1998.