Security management architecture for access control to network resources

G. Prem Kumar
P. Venkataram

Indexing terms: Network security management, Access control techniques, User access control, Access control to network resources

Abstract: Threats to network resources increase exponentially with the growth of the network/users and the technological developments. In the paper, the authors describe a security management framework for access control to the network resources. They deal with this in three steps. In the first part, user access control to a given network is discussed. In the second part, access control to the network resources, both by self-usage and delegation, revocation of delegation is presented. The third part explains the access control to the resources belonging to the other networks. A neural network based model is developed for intrusion detection to overcome most of the limitations of the existing systems.

1 Introduction

As the network resources and their availability increase, so does the threat for the resources to be misused by both legitimate and unauthorised users. From the network management point of view, security management [1–3] plays a vital role in maintaining the privacy and integrity of the network. The types of threats in access control to network resources range from masquerade (or false claim of the origin), illegal associations, nonauthorised access, denial of service, repudiation (or false denial of performing the task) and Trojan horses. Two major ways of solving the security management problem are preventative measures by employing strong underlying security architecture, and reactive measures by using intrusion detection techniques.

This paper focuses on the preventative measures of security management by proposing efficient security protocols and on reactive measures by developing a neural network based intrusion detection model. The proposed preventative approach is described in three parts (see Fig. 1). The first part of it is the user access control to the network (UA-protocol), which deals with providing access to the network for authorised users. This part is mainly concerned about user login from any host in the network to any other host within the network. In the second part, access control to the services is modelled as a client-server model (CS-protocol). In the CS-protocol, each of the services offered by the network has been modelled as a specific server and the user approaches to them as a client. Access control mechanism is developed for this class of services is further divided into two parts: a legitimate user directly accessing the service and a legitimate user delegating other user(s) of the network to access the service. We also consider revocation of the delegation. The third part of the proposed preventative approach defines the secure access of the resources belonging to other networks/domains (interdomain protocol or ID-protocol). In the proposed reactive approach, the information in the security management information base (SMIB) is used to identify the intrusion that may take place and alert the security manager of the possible event. A neural network based on the backpropagation is suitably tuned to devise the intrusion detection system (IDS).

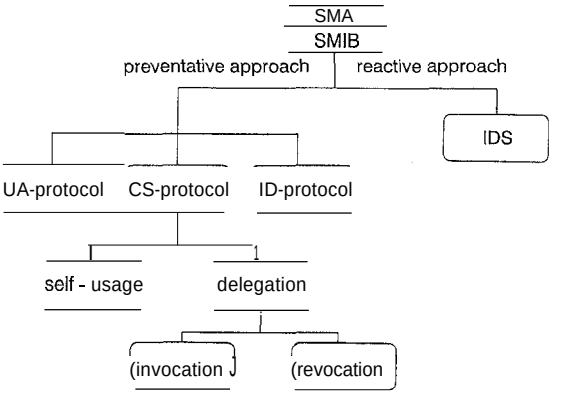


Fig. 1 *Proposed model for network security architecture*
SMA = security management architecture
SMIB = security management information base
UA-protocol = user access protocol
CS-protocol = client-server protocol
ID-protocol = interdomain protocol
IDS = intrusion detection system

2 User access control

Access control mechanisms, which are extensively being tried, could be either one-way or two-way authenticated. Access control involves two entities (i) identify and (ii) identifier. Identity is the user who approaches the access control system and claims to be an authorised user of the system. Identifier is the access control system that authenticates the identity after proper verification. In one-way authentication mechanism, the

identity has to provide the proof to the identifier that he/she is the authorised user. On the other hand, in two-way authentication, both the identity and identifier have to prove to one another their identity before access to the device is permitted.

2.1 One-way authentication

Access control to the Unix kind of system uses one-way authentication mechanism by means of passwords. The user supplies the password along with his/her login identification (id) during the login. The system then compares the one-way encrypted password with the stored password and the successful match permits the login. A drawback with this kind of system is that there is no method to judge whether the user has logged into the intended host or not.

2.2 Two-way authentication

In the two-way authentication mechanism, the identifier poses a challenge to the identity. If the identity gets convinced with the challenge and trusts the identifier, it returns a response. On a successful match with the expected response, the identifier gives access to the identity. This method is also known as challenge-response pair access control. A smart card can be used to have encrypted and lengthy challenge/response [4]. On each successful attempt, the sequences in challenge-response pair are updated at both the identity and the identifier. Although the challenge-response pair of authentication mechanism seems to be foolproof, it has its own limitation in that any user who possesses the srnart card can have access to the device without further verification.

To overcome the problems faced by the above schemes and provide a higher level of security, we propose a hybrid method for user access to the hosts in a communication network.

2.3 Proposed method for user access control

We consider a communication network as a 6 tuple, (HS, LK, U, LS, S, SM) where *HS* is a set of hosts, *LK* is a set of links connecting the hosts, *U* is a set of users (grouped hostwise), *LS* is a login server (one LS per each host or only one for the entire network), *S* is a set of servers like print server(s), mail server(s), file server(s), and *SM* is the security manager to deal with key distribution and local/inter network security issues.

At the time of login creation, each user is given a smart card and a password which can be changed on any successful login. The smart card has a sequence of words, called challenges, whose copies are held sequentially with the login server. The smart card has a private key of its own and a public key of the login server, counterparts of which are held with the LS. We intend to use the method of public key cryptography for a higher level of security, details of which can be found in [5, 6].

Whenever a user *u* attempts to login, through a host *h* to the registered host *H*, the user provides *id* and *H* to host *h*. Then smart card (SC) throws a signed challenge, encrypted with public key of LS, to the host *h*. The host *h* signs and encrypts the message and forwards it to the LS on host *H*. The *LS*, on verifying the host *h* and the smart card, reads the challenge and compares with the expected challenge of that user. If the verification succeeds, the *LS* signs and encrypts the challenge, along with the one-way encrypted password of the user. On successfully comparing the challenge, the smart card prompts the user for password. If the user password matches with the one sent from the *LS*, the smart card sends a (signed and encrypted) message to the *LS* (through *h)* to create a login connection and concludes the access control phase successfully (see Fig. 2).

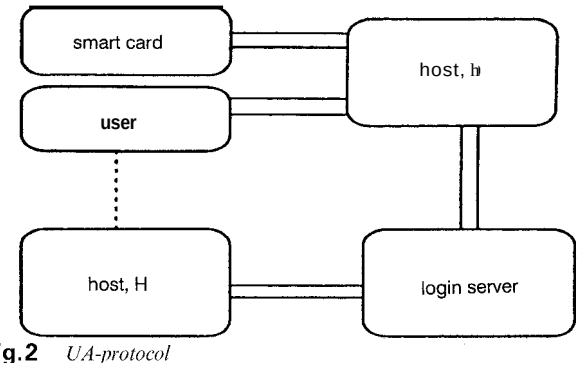


Fig. 2 *UA-protocol*

Note that, in the proposed model, the traditional challenge-response pair access control mechanism is reversed. In other words, the identity (user) first poses the challenge to the identifier (login server), since *LS* does not know who is the next user going to login. We call this UA-protocol access control mechanism as challenge-password pair (CP-pair) based approach.

Notution:

(i) '$a \rightarrow b : \{p_1, ..., p_n\}$' indicates that *a* is conveying a set of parameters $\{p,\}$ to b, where p_i may be one of messages, addresses, challenge, request, permission, id, address of the host, password etc.

(ii) 'a : Action' indicates that *a* is performing Action.

(iii) $\text{Message}_{X^{-1}\,Y}$ indicates that the message is encrypted by the public key of *Y* and signed by *X* using the private key of *X*. For more details on the syntax used to describe the protocols in this paper, refer to [7].

UA-protocol

(i) u $\rightarrow$ h : {id, *H*}

(ii) SC $\rightarrow$ h : $\{\langle \text{challenge} \rangle_{SC^{-1},LS}\}$

(iii) h $\rightarrow$ LS : $\{[\text{id, H, } \langle \text{challenge} \rangle_{SC^{-1},LS}]_{h^{-1},LS}\}$

(iv) LS : Decrypts and verifies identity of *h* and SC. Then compares the *challenge* with the stored. If the challenge matches, returns the one-way encrypted password. Otherwise, sends an *abort* message

(v) LS $\rightarrow$ h : $\{[\text{id, H,}\langle \text{challenge, one-way encrypted password} \rangle_{LS^{-1},SC}]_{LS^{-1},h}\}$

(vi) h $\rightarrow$ SC : $\{ \langle \text{challenge, one-way encrypted password} \rangle_{LS^{-1},SC}\}$

(vii) SC : Decrypts the message from LS. If the *challenge* matches, the SC prompts the user for password by concluding that the host *h* is trustworthy. Otherwise, sends an *abort* message

(viii) u $\rightarrow$ SC: *{password}*

(ix) *SC* : One-way encrypts the password and if it matches with the one sent by the LS then updates its challenge sequence else *uborts*

(x) SC $\rightarrow$ h : $\{\text{id, H, } \langle \text{challenge, 'CREATE'} \rangle_{SC^{-1},LS}\}$

(xi) h $\rightarrow$ LS . $\{[\text{id, H, } \langle \text{challenge, 'CREATE'} \rangle_{SC^{-1},LS}]_{h^{-1},LS}\}$

(xii) LS : Creates a login connection for *u* at *h* and updates the corresponding challenge sequence

2.4 Analysis of the proposed model

Now, we turn to analyse the proposed protocol.

(i) The length of the challenge posed by the smart card is generally longer, and thus avoids easy guessing.

(ii) By using the two-way authentication, both the identity and the identifier are assured that the other is legitimate.

(iii) Replacing the automated response part of the challenge-response pair mechanism by password:

(a) will not make any person whoever carries the smart card as a legitimate user unless helshe knows the password. This avoids stealing of smart cards.

(b) reduces the hardware meant for response sequence in the smart card and the memory usage with the login server. Instead, the password is remembered by the user and an encrypted version of the password is stored at the LS.

(iv) If a finite number of challenges are used, the sequence repeats after all the challenges are posed which may lead to guessing of the challenges. This can be avoided by a method proposed in [8] where the used challenges are modified indefinitely at both identity and identifier in the same fashion.

(v) Since the challenges are nonrepeating, 'challenge' in Step 5 of the UA-protocol avoids replay of old messages.

(vi) The use of signed and encrypted messages ensures privacy and integrity of the message [9].

(vii) The use of passwords along with the challenges doubly authenticates the legitimate users' possession of the smart card.

(viii) LS verifies the trustworthiness of the host h, in Step 4 of the protocol.

(ix) The case of group-login, where the same **id** is shared by many users, can be handled by maintaining as many challenge sequences with the LS as the number of smart cards issued for that **id.** The group-login **id** in the protocol is defined as '*id*; *#user*' where #user is the number given to the group-user.

(x) UA-protocol can be applied to distributed systems. A login to a distributed system enables the user to access the entire system of hosts without having to specify the registered host H.

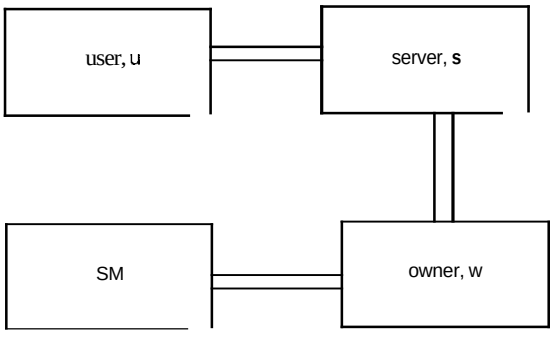
(xi) Coming to the performance measures of the proposed protocol, probability of false rejection is absolutely zero as it is an electronic access control system (assuming a lossless underlying network), while the false acceptance ratio is almost zero since it is tedious to send a signed-encrypted message twice to the LS. (Breaking the secrecy of public key cryptography is notoriously difficult *[5]* and doing this process twice further reduces the possibility.)

(xii) All the information, at different stages of the protocol, regarding any attempt to illegal access is saved in the security management information base.

3 Client-server protocol

The next part of the preventative approach is to ensure that the legitimate users only allowed to access various services within the network. We refer to this as a client-server model.

We classify client-server model into two categories: (i) a legitimate user directly approaching the server,



3.1 Client-server: self-usage

The proposed client-server model is shown in Fig. **3.** A user u generates a client process c to get a service by a server s, owned by **w.** The servers can be owned by a single owner or multiple owners and there may be transfer in ownership of the server but this transfer is transparent to the client. It is enough if the server keeps track of its owner(s).

The proposed access control model is based on the public key cryptography where each of the servers possess the public and private keys like any other user. Whenever a client process c is generated to get work done from a server s, the request with specific requirements is forwarded through the host. The server, on receiving the client's request, forwards it to its owner w for the permission. The owner of the service, depending on the security measures and the capacity of the server, may give 'Permission' to access the service in the form of a *ticket.* The ticket is issued to the server if it is a one-time job or to both: the user and the server, if it has to be used more than once. In the former case, the server will directly render the service without informing the user and in the later case, the client gets the service on producing the ticket on every occasion.

The protocol proceeds as follows:

CS-protocol

(i) u $\rightarrow$ s : {(Request),-I,,} where *Request* = {nonce, period, #times, Personal/Proxy, List of Proxies, Any other options } and *nonce* is the *id* of the request.

(ii) s $\rightarrow$ w : $\{\langle \text{Request} \rangle_{s^{-1},w}\}$

(iii) w : May allow the use of service or reject the request. If the service does not involve bulk transfer of data, GOESTO Step 6 (a). Otherwise, procures the session key from *SM* and sends it to both server and the user along with the ticket

(iv) w $\rightarrow$ SM : $\{(\text{nonce}, \text{Session\_Key\_Request}\rangle_{w^{-1},SM}\}$

(v) SM $\rightarrow$ w : $\{(\text{nonce}, \text{Session\_Key}\rangle_{SM^{-1},w}\}$

(vi)(a) w $\rightarrow$ s : $\{(\text{Permission}, \text{Session\_Key}\rangle_{w^{-1},s}\}$

Table 1: Table maintained by the server of the service during CS-protocol

User	Nonce	Time duration	No. of times	Proxy/personal	Proxy list	Any other option
James	34589	t1–t2(absolute)	2	Proxy	Joy, Joe, Mary	Urgent
Mary	34690	t5–t6(absolute)	1	Personal	–	–

(b) w : If the *Request* is a one-time request, GOESTO Step 7.

(c) $w \rightarrow u$: $\{[\langle \text{Permission} \rangle_{w^{-1},s}, \text{Session\_Key}]_{w^{-1},u}\}$ where Permission = {u, nonce, period, #times, Personal/ Proxy, List of Proxies, Any other options} and #times is the number of times the service can be offered.

(vii) s : If the *Request* is one-time request then renders the service and GOESTO Step 9. Otherwise, stores the *Permission* and keeps track of it as shown in Table 1.

(viii) $(u \rightarrow s)^n$: $\{[\langle \text{Permission} \rangle_{w^{-1},s}, \text{Current\_Request}]_{u^{-1},s}\}$ where the format of *Current–Request* is similar to that of *Request* indicates its restricted usage and $(u \rightarrow s)^n$ indicates *n* messages from *u* to *s*.

(ix) (a) s : Renders the service $\{\langle \textit{Reply} \rangle_{s^{-1},u}\}$ and GOESTO Step 10 if it is a one-time request or when the limit exceeds.

(b) s : Updates the Table 1 and further keeps track of it by awaiting further requests starting from Step 8.

(x) $s \rightarrow w$: $\{[\langle \text{Request} \rangle^n_{u^{-1},s}]_{s^{-1},w}, \text{Service\_Rendered}\}$ where $\langle \textit{Request} \rangle^n$ indicates *n* requests served by s.

A few points about the protocol are worth mentioning.

(i) Apart from several intricate details that are incorporated in our protocol, it differs from Kerberos [12], in the following ways:

(a) Instead of timestamp, nonce (a unique identifier) is used and the period is only specified to represent the permissible usage period subjected to other constraints;

(b) The ticket is issued only when the client is going to use the request more than once or when it is used for delegation; and

(c) The delegation mechanism is generalised.

(ii) In the protocol, the nonce contained within the Request is used as *id* of that process, which is sufficiently long and thus cannot be easily reproduced. If the nonce already exists with the server, a different number is substituted and the new nonce is communicated to the client. (The idea here is to reuse the nonce as the id of the request since it is a unique and long number.)

(iii) The nonce also serves to avoid the replay of old messages.

(iv) The period field in the Request is the time during which the client is intending to get the service. During that period, the owner of the service may accept allowing the client, maybe with some other restrictions. Table 1 specifies one such access control list (ACL).

(v) The period field in the Request specifies the time duration in which the client can get the service. During the period, server accepts the client's request(s) and maintains it in access control list during the process (as shown in Table 1).

(vi) If the period is over or the service limit is exceeded, the request is revoked (as discussed in Section 3.3), the entry in the ACL is removed and the same will be notified to the user and the proxies.

3.2 CS-protocol: delegation

The protocol provides the access to the illegitimate users of the service through the legitimate users. The basic philosophy is to delegate the work to the illegitimate users. Here, the scheme incorporates the capability mechanism by means of delegation.

We propose to use chained delegation. The legitimate user *u*, who has the permission from the owner of the service *w* to use the server *s*, delegates the task to a delegate *d*.

The protocol proceeds as follows:

(i) $u \rightarrow d$: $\{s, \text{'Proxy'}, [\langle \text{Permission} \rangle_{w^{-1},s}, \text{Optional\_Session\_Key}]_{u^{-1},d}\}$

(ii) $d \rightarrow s$: $\{u, \text{'Proxy'}, [\langle \text{Permission} \rangle_{w^{-1},s}, \text{Current\_Request}]_{d^{-1},s}\}$

(iii) s : Renders the service to the delegate as in Step 9 of the *client-server: self-usage* protocol.

The server verifies the Proxies list and permits only authorised delegates.

3.3 Revocation of delegation

It may be necessary for the user to revoke a delegation. The protocol provides the revocation procedure for the purpose.

(i) $u \rightarrow s$: $\{d, \text{'Revoke'}, [d, \text{nonce}, \langle \text{Permission} \rangle_{w^{-1},s}, \text{Revoke\_Request}]_{u^{-1},s}\}$ where Revoke_Request = {nonce, Reduced/Removed permissions to delegate *d*}.

(ii) s : Updates the entry corresponding to the delegation in ACL and intimates the delegate of the same.

4 Interdomain protocol

The interdomain protocol (ID-protocol) uses either hierarchical or distributed security managers to provide access control to the services belonging to the other networks. It is proposed that the security managers of the networks have a way of communicating messages in a secure manner among themselves. This could again be achieved by using the public/private keys of each of the SMs. Any service access that belongs to a different network can be requested by the SM of the local network after appropriate authentication of the user or its client process. Alternatively, any request to access the remote resource should first exchange the keys and then use the client-server: self usage protocol, as described in Section 3.1. But, we prefer to use the SM to interact with the other domains as the process overhead is higher in the second method and each client/ user has to exchange the public/private keys with the servers of the other machines and manage them independently.

A request or a message that has to be securely sent from a client in one network to the server in the other will start by sending a request to the former's **SM** to forward it to the remote server. The local SM (LSM) then communicates with the remote SM (RSM), in a suitable way, and requests the service. RSM then runs it as a client with the server as though it is a local

request and gets back the reply on the reverse path. The protocol is shown in Fig. 4.

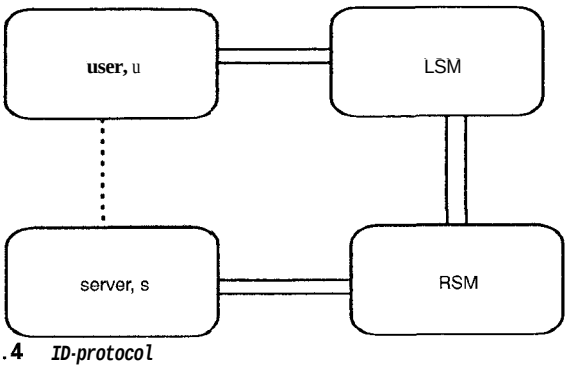


Fig. 4 *ID-protocol*

The protocol proceeds as follows:

ID-protocol

(i) u → LSM : {'Remote Request', s, RSM, $\langle \text{Request} \rangle_{u^{-1},LSM}$}

(ii) LSM → RSM : {'Remote Request', s, $\langle \text{Request} \rangle_{LSM^{-1},RSM}$}

(iii) The service is accessed replacing, *u* by RSM, in *Client-server: self-usage* protocol.

(iv) RSM → LSM : {'Remote Request Reply', $\langle \text{Reply} \rangle_{RSM^{-1},LSM}$}

(v) LSM → u : {'Remote Request Reply', $\langle \text{Reply} \rangle_{LSM^{-1}u}$}

5 Intrusion detection system

As a part of network management, network intrusion detection has received lot of attention. In this Section, the second part of the proposed security management architecture, the reactive approach, namely, intrusion detection system is discussed. We briefly describe the existing methods for intrusion detection and then present a new approach using a neural network.

5.1 Prior works

Since one cannot expect any networked system to be foolproof in spite of so many protocols developed to keep the resources secure, there is always a necessity to develop an intrusion detection system that identifies a possible intrusion even before it takes place and stop such an event [2, 13].

Statistical methods have been used extensively in intrusion detection systems [1]. In this approach, the user behaviour over time is observed and if the standard deviation exceeds the expected value (or mean), it is identified as an intrusion. State transition analysis is carried out to identify the intrusions in [14]. In the state transition method, the series of events that would cause an intrusion are represented as states. An intrusion is reported when the specified events take place in that particular order until the final state. Petrinets are used for state transition analysis in [15] for intrusion detection.

AI techniques have been widely used in intrusion detection systems [16]. Rule-based systems have been used for intrusion detection [1, 17]. In rule based systems, the possible intrusions are coded as 'if-then' rules. When all the conditions in the 'if part of a rule are matched, 'then' part of the rule is 'fired' by raising an intrusion. Intrusion detection based on the time interval between the successive keystrokes while typing a known sequence of characters is suggested by using a multilayer neural network system in [18] and that based on fuzzy algorithms in [19]. Few researchers have attempted to design a neural network based IDS to predict every next command of the user [20]. This technique classifies the unpredicted command as an intrusion and reports to the security manager to initiate necessary action.

5.2 Observations from the existing models

A careful study of the existing models on intrusion detection suggests that:

(i) use of statistical methods to analyse the intrusion on the exhaustive data is very time consuming for both the intrusion detection system and the intrusion detection expert.

(ii) since the users have the tendency to learn or try out new commands from time-to-time, it may be difficult to predict a next command in their usage of the network. Adaptation of user fancy and style in the prediction system is a complex task.

(iii) the use of key stroke, based on speed or pressure, matching for intrusion detection is limited to the user login in most of the cases. The user speed/pressure to enter the commands may vary over time which then may raise false alarms.

(iv) in case of state transition method, if the events do not take place in sequence, the intrusion cannot be foreseen until it reaches in the last state of the intrusion. If all such combinations have to be included in the state transition diagram, it leads to state explosion.

(v) in the case of co-operating users or the same user making an attempt for intrusion at different times in various steps, it may be difficult to identify the intrusion unless the information is collected globally and tracked.

(vi) hard coding of rules using rule-based systems is not advisable since no intrusion detection system can be assumed to be complete and there is always a necessity to learn new intrusions (rules).

6 Proposed model for intrusion detection

To overcome some of the limitations in the existing models on intrusion detection, we propose a new approach for IDS that uses a neural network model based on the backpropagation learning algorithm.

6.1 Formal description of the system

A general IDS system can be described to comprise the users U = $\{u_1, u_2, ..., u_m\}$, the set of commands executed by the users C = $\{c_1, c_2, ..., c_n\}$, a finite set of possible intrusions that can take place I = $\{i_1, i_2, ..., i_p\}$, the recognised intrusion sequences R = $\{r_1, r_2, ..., r_q\}$ and unrecognised intrusion sequence N = $\{n_1, n_2, ..., n_s\}$.

Further, we classify the commands into two categories. The first category consists of the commands (C_1) that a single user carries out to cause an intrusion.

$$C_1 = \{u_i(C)\} \tag{1}$$

where $u_i(C)$ is the set of commands that user u_i has executed, out of the commands C.

The second category comprises the set of commands carried out by co-operating users (C_2) which can lead

to the intrusions. They are collected on the global basis from the **SMIB** and not based on the individual user.

$$C_2 \subset C$$

Every intrusion is either reported from the recognised set of intrusions or it has to be brought out from yet unrecognised set of intrusions.

$$I = R \cup N \tag{2}$$

In other words,

$$I = \{g(\{c_h\}) | ((\{c_h\} \subset C1) \vee (\{c_h\} \subset C2)) \wedge ((g(\{c_h\}) \in R) \vee (g(\{c_h\}) \in N))\} \tag{3}$$

where g(.) is a function that associates a set of commands $\{c_h\}$ to a set of intrusions.

6.2 Approach

The proposed neural network based IDS model is developed in two steps. In the first step, the expert knowledge is captured offline in the form of a set of rules and then in the second, based on the principle of induction, the system is allowed to work online to learn any possible unencountered intrusions. The order in which the events have taken place has least significance in the NN-based systems. Thus the events of co-operating users and the events carried out by a single user at different instances of time are handled together to foresee the intrusion. The rule-based system, implemented as a connectionist (a neural network) model, minimises the search.

6.3 Neural network model

The neural network model devised for intrusion detection is based on the backpropagation model [21, 22] (see Fig. 5).

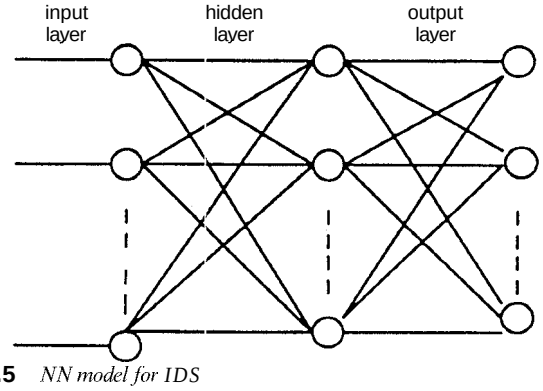


Fig.5 *NN model for IDS*

The neural network has an input layer that is designed to accept the binary inputs, representing the events of interest (commands $\{c_j\}$ of users $\{u_k\}$) those have appeared in the **SMIB.**

The output layer of NN is expected to indicate the possible intrusions.

The NN also has one or more hidden layers depending on the complexity of the problem (i.e. approximate number of events of interest, number of rules, and the number of intrusions). The number of neurons in the hidden layer also depend on the estimate of the number of the exemplars provided for training [23].

Each layer of the NN consists of one or more neurons. The output of one layer is fed as the input of the next. The neurons in one layer are connected to those in the next with an adaptive weight. The net input to the neuron is (h_i).

$$h_i = \sum_j (w_{ji} * input_j) \tag{4}$$

where w_{ij} is the weight of the link connecting neuron j in the preceding layer to the neuron i in the current layer and $input_j$ is the input from the neuron j. The input to bias neuron in the preceding layer is set to –1 and the weight is adjustable. Essentially, the weight on the bias link in the previous layer indicates the threshold of each neuron in the current layer.

The output of each neuron is determined by applying a transfer function $f(.)$ to the net input to the neuron. We use the activation function:

$$f(x) = (1 + e^{-x})^{-1} \tag{5}$$

The training of the neural network is done in two passes. The forward pass is used to evaluate the output of the neural network for the given input with the existing weights. In the reverse pass, the difference in the neural network output with the desired output is compared and fed back to the neural network as an error to change the weights of the neural network.

In the reverse pass, suppose for the particular neuron i in output layer L, the output is y_i^L while the desired value is d_i^L. The backpropagated error δ_i^L is:

$$\delta_i^L = f'(h_i^L)[d_i^L - y_i^L] \tag{6}$$

where h_i^L represents the net input to the ith neuron in the Lth layer and $f'(.)$ is the derivative of $f(.)$.

For layers $l = 1, ..., (L-1)$, and for each neuron i in layer l the error is computed using:

$$\delta_i^l = f'(h_i^l) \sum_j (w_{ij}^{l+1} * \delta_j^{l+1}) \tag{7}$$

The adaptation of the weight of the link connecting neurons i of layer l and neuron j of layer $(l-1)$ is done using:

$$w_{ij}^l = w_{ij}^l + K * \delta_i^l * y_j^{l-1} \tag{8}$$

where K is the learning constant that depends on the rate at which the neural network is expected to converge.

This neural network algorithm essentially minimises the mean square error between the neural network output and the desired output using gradient descent approach.

The criteria for the neural network to turn from offline mode to online mode can be one of: (i) when the desired output is same as the neural network output for all the training patterns; (ii) when the mean square error between the neural network output and the desired output is less than the specified threshold (for example, when the mean squared error per unit output is less than 0.01); (iii) when the gradient is sufficiently small (by definition, the gradient will be zero at the minimum).

Algorithm intrusion detection

Begin

(i) Construct a neural network with 1, 2, ..., L layers. And in each layer select number of neurons that suits the problem [23].

(ii) Set the input of the bias neuron in each layer to –1.0.

(iii) $w_i = \delta \forall i, j$, a random value in the range (0, 1).

(iv) Choose an input pattern from the set of exemplars if the **IDS** is in offline mode. Otherwise, take events of interest from SMIB as the input.

(v) Propagate the signal forward through the neural network until the output layer.

(vi) For each of the neuron i in the output layer (layer L), compute the error:

$$\delta_i^L = f'(h_i^L)[d_i^L - y_i^L]$$

(vii) For $l = 1, ..., (L - 1)$, and for each neuron i in layer l compute the error

$$\delta_i^l = f'(h_i^l) \sum_3 (w_{ij}^{l+1} * \delta_j^{l+1})$$

(viii) Update the weights using:

$$w_{ij}^l = w_{ij}^l + K * \delta_i^l * y_j^{l-1}$$

where K is the learning rate.

(ix) If the error in the output layer ($\Sigma_i[d_i^L - y_i^L]^2$) is within permissible limits, switch from offline mode to online mode of operation.

(x) Repeat by going to Step 4, till IDS is active.

End

7 Experiment with IDS

We demonstrate the features of the proposed IDS with an example.

Consider a model where the events are collected for the following three cases:

(i) during the user login

(ii) access to network resources

(iii) access to interdomain resources.

The input corresponding to the first layer neuron is set to '1' if the event is present and to '0' otherwise.

The IDS considers the following events:

(i) rlogin/telnet/ftp is denied for remote connections.

(ii) rloginitelnetiftp is denied since remote host is not permitted as directed by /etc/ hosts.deny

(iii) rlogin/telnet/ftp is denied. (Here, remote host not denied but the user intrusion attempt has failed.)

(iv) (from root account) *cp /bin/sh usrlroot* is executed. (*root* shell is copied to user area.)

(v) (from root account) *chmod 4755 usrlroot* is executed. (User is permitted to use *root* shell.)

(vi) *usrlroot* is executed. (User is running the *root* shell.)

(vii) command *cd* to change directory to unrelated groups.

(viii) attempt is made to delete the files of other users/ area.

(ix) *su* succeeded.

(x) *su* failed.

(xi) printifax is denied to the user.

(xii) printifax is denied to the user as the limit exceeded their permitted values.

The intrusions are:

(i) remote host attempting a connection.

(ii) unauthorised user of an unauthorised system is attempting to login.

(iii) root shell is copied.

(iv) root shell is copied and executed.

(v) with proxy root shell, directory is changed.

(vi) with proxy root shell, files are deleted.

(vii) *su* succeeded, trace subsequent commands.

(viii) *su* failed, identify the user and the terminal that initiated *su*.

(ix) *user* is denied to access *print/fax*.

(x) *user* printifax limit is exceeded, initiate the appropriate measures.

Table 2 describes 16 events and 10 intrusions of the IDS with the initial knowledge (in the form of 12 rules) in a public network.

The commands that fall in the co-operating user intrusion are the events 4–6. The usage of these commands is collected globally from SMIB and not from each user's record. Where as the other events (1–3 and 7–12) are gathered user-wise and fed as a pattern to the neural network to evaluate the possibility of any intrusion.

Table 2: Initial knowledge used by the IDS

S. no.	Events	Intrusions
1.	1110000000000000	1100000000000000
2.	1100000000000000	1000000000000000
3.	1000000000000000	1000000000000000
4.	0001000000000000	0010000000000000
5.	0001100000000000	0010000000000000
6.	0001110000000000	0001000000000000
7.	0001111000000000	0001100000000000
8.	0001110100000000	0001010000000000
9.	0000000010000000	0000001000000000
10.	0000000001000000	0000000100000000
11.	0000000000100000	0000000010000000
12.	0000000000010000	0000000001000000

Table 3: New rule

S. No.	Events	Intrusions
13.	0000000000011000	0000000000010000

7.1 Illustrative example

Consider S. No. 6. of Table 2. The events 4, 5, 6 i.e. *(cp /bin/sh usrlroot), (chmod 4755 usrlroot)* and *(usr/root)* are set to '1' and the other events are set to '0'. This raises an intrusion 4: *root shell is copied and executed.* Further, the events specified in **S.** No. 7 and 8 have more than one intrusion. In case of **S.** No. 7, the events 4, 5, 6 and 7 raise intrusions 4 and 5 whereas the events 4, 5, 6 and 8 in **S.** No. 8 raise intrusions 4 and 6.

7.2 Inclusion/deletion of rules

Let us suppose that the IDS encounters a set of events that requires a new set of intrusions to be raised. Since the neural network is being trained in online mode, such a combination can be incorporated as a new rule if it occurs often enough. Similarly, if an existing rule becomes invalid, the neural network unlearns it overtime.

A neural network model with a few spare input and output neurons can accommodate new events/intrusions. As shown in Table 2, although there are 12 events and 10 intrusion, we have chosen a neural network with 16 input neurons, 15 output neurons and 6 hidden neurons. To illustrate the inclusion of a new rule, consider the rule given Table 3.

In the new rule, a new event 13 is added to the existing event 12. This results in a new intrusion 11

Event **13:** *print/fax* is issued too many times

Intrusion 11: Trace the user

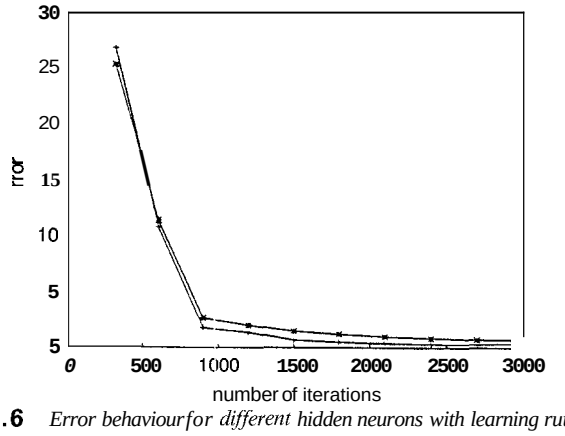


Fig. 6 *Error behaviour for different hidden neurons with learning rutc = 0.1*
+ six hidden neuron
* four hidden neuron

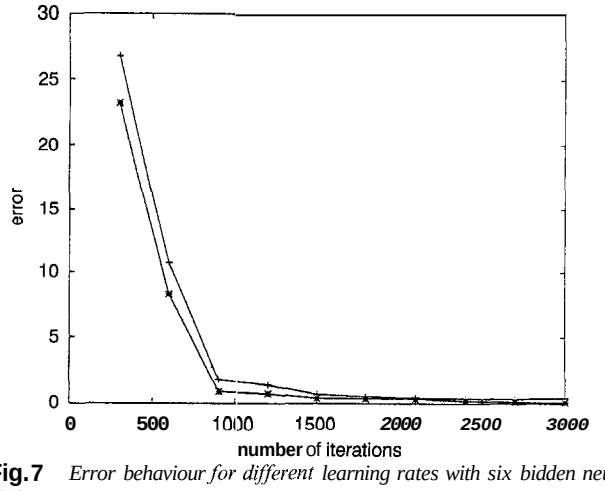


Fig. 7 *Error behaviour for different learning rates with six bidden neurons*
+ learning rate = 0.1
* learning rate = 0.2

7.3 Results

The neural network is devised with 16 neurons in the input layer and 15 neurons in the output layer. The neural network model is developed in C++ programming language on a Sun Sparc 20 workstation, running SUN OS 4.2. The hidden layer is chosen with varying number of neurons. Similary, different learning rates were incorporated to train the neural network. For illustration, we consider a neural network convergence with the following parameters:

(i) with four and six hidden neurons;

(ii) with learning rates 0.1 and 0.2.

In each case, the error in the neural network for the above parameters is measured while presenting the exemplars given in Table 2. The error computed is the sum of the squared error of each of the output neurons for all the 12 possible rules at iteration instances in multiples of 300. It is observed that the neural network started converging after a total 800 iterations of the exemplars.

In the case of the neural network with alearning rate of 0.1 and with six hidden neurons shows better performance over the one with four hidden neurons for the same learning rate (i.e. less error) after 500 iterations (see Fig. 6).

Similarly, an experiment is conducted for different values of learning rates (0.1 and 0.2) when the hidden neurons are six (see Fig. 7). The graph shows that the neural network with learning rate of 0.2 converges much faster than the one with 0.1. But the learning rate in general is a compromise chosen by the rate at which the neural network is expected to converge and also to avoid the noise in inputting of the desired output.

8 Conclusion

A comprehensive network security management architecture is developed in two parts, namely, preventative approach and reactive approach. In the first, an attempt is made to improve the robustness of the protocols. In the second, a neural network based intrusion detection system is developed. The model is analysed thoroughly and the results are impressive [24].

9 Acknowledgments

The authors thank the anonymous referees for their comments to improve the content of the paper.

10 References

1 DENNING, D.E.: 'An intrusion-detection model', *IEEE Trans.*, 1987, **SE-13,** pp. *222–232*
2 MUKHERJEE, B., HERBERLEIN, L.T., and LEVITT, K.N.: 'Network intrusion detection', *IEEE **Network,*** 1994, pp. 26–41
3 O'MOHONY, D.: 'Security considerations in a network management environment', *IEEE Network,* May/June 1994, pp. 12–17
4 SHERMAN, S.A., SKIBO, R., and MURRAY, R.S.: 'Secure network access using multiple applications of AT&T's smart card', *AT&T Tech. J.,* 1994, **73,** (5), pp. 61–72
5 DIFFIE, W., and HELLMAN, M.: 'New directions in cryptography', *IEEE Trans.*, 1976, **IT-22,** (6), pp. 644–654
6 RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: **'A** method for obtaining digital signatures and public-key cryptosystems', *Comm. ACM,* 1978, **21,** (2), pp. 120–126
7 BURROWS, M., ABADI, M., and NEEDHAM, R.M.: 'A logic of authentication', *ACM Trans.,* 1990, **CS–8,** (1), pp. 18–36
8 SWAMY, V.C.J.: 'Electronic access control system: a new approach'. MSc (Eng.) dissertation, Department of Electrical Communication Engineering, Indian Institute of Science, 1994
9 WOO, T.Y.C., and LAM, **S.S.:** 'Authentication for distributed systems', *IEEE Cornp.,* 1992, ***25,*** (1), pp. 39–52

10 NEUMAN, B.C.: 'Proxy-based authorization and acounting for distributed systems'. IEEE Computer Society symposium on *Security and privacy,* 1993, pp. 283–291
11 VARADHRAJAN, V., ALLEN, P., and BLACK, S.: 'An analysis of the proxy problem in distributed systems'. IEEE Computer Society symposium on *Security and privacy,* 1991, pp. 255–275
12 STEINER, J.G., NEUMANN, B.C., and SCHILLER, J.J.: 'Kerberos: an authentication service for open network systems'. Proceedings of winter USENIX conference, 1988, pp. 191–201
13 ESMAILI, M., SAFAVI-NAINI, R., and PIEPRZYK, J.: 'Intrusion detection: a survey'. Proceedings of 12th international conference on *Computer communications,* Seoul, Korea, August 1995, pp. 409–414
14 ILGAN, K., KIMMERER, R.A., and PORRAS, P.A.: 'State transition analysis: a rule-based intrusion detection approach', *IEEE Trans.,* 1995, **SE-21,** (3), pp. 181–199
15 KUMAR, S., and SPAFFORD, E.H.: 'A pattern matching model for misuse intrusion detection', (Submitted for publication)
16 FRANK, J.: 'Artificial intelligence and intrusion detection: current and future directions', (Submitted for publication)
17 BAUER, D.S., and KOBLENTZ, M.E.: 'NIDX – An expert system for real-time network intrusion detection'. Proceedings of IEEE *Computer network security* symposium, 1988, pp. 98–106
18 OBAIDAT, M.S., and MACCHAIROLO, D.T.: 'A multilayer neural network system for computer access security'. *IEEE Trans.,* 1994, **SMC-24,** (5)
19 HUSSIEN, B., MCLAREN, R., and BLEHA, S.: 'An application of fuzzy algorithms in a computer access security system', *Patt Recog. Lett.,* 1989, **9**, pp. 39–43
20 DEBAR, H., BECKER, M., and SIBONI, D.: 'A neural network component for an intrusion detection system'. Proceedings of IEEE symposium on *Research in computer security and privacy,* 1992, pp. 240–250
21 LIPPMAN, R.P.: 'An introduction to computing with neural nets', *IEEE ASSP Mag.,* April 1987, pp. 4–22
22 RUMELHART, D.E., HINTON, G.E., and WILLIAMS, R.J.: 'Learning internal representations by error propagation', *in* RUMELHART, D.E., and MCCLELLAND, J.L. (Eds.): 'Parallel distributed processing: explorations in the Microstructure of Congnition, Vol. 1: Foundations' (MIT Press, 1986)
23 BARTO, A.G., SUTTON, R.S., and ANDERSON, C.W.: 'Neuronlike elements that can solve difficult learning control problems', *IEEE Trans.,* 1983, **SMC-13,** pp. 835–846
24 KUMAR, G.P.: 'Integrated network management using extended blackboard architecture'. PhD