Secure Connection for Private Local Area Network

Anoar Abbas htaab
AL-Mustanseryia university

Abstract:

With current concerns over computer security, particularly related to network intrusions, misuse, and even cyber-terrorism, there is a need for tools to monitor and analyze systems to identify when an attack is occurring.[1]

This paper represents the design and implementation of secure connection for private Local Area Network (LAN)by build Intrusion Detection System (IDS) depend on the rule-based technique for detection the intruder and to generate the rules of detection extract from the collection of analysis and information of log file and events in the network. The rule based intrusion detection system is able to learn and improve their rules with reaction of the network events, four techniques were presented to work in the IDS. The first is the packet filtering that was used to prevent unauthorized user from pass through network ,second the rule-based inference engine used to generate rules and for detection intrusions. The third technique is the session analysis that used to detect the session between users and the server, fourth is the watermark in the authorized user packet. This technique help the system to detect the unauthorized user in the start it work and reduce the error or the intrusion attacks.

Many attack types especially in the TCP protocol were detected with high resolution. Finally, we must say that the rule–based intrusion detection system give better detection efficiency from the statistical intrusion system for many types of attack[2].

1. Introduction :

The proposed system operation started by initializes the network services and domain of the network for controlling the network and services. After initialization stage, the proposed system start the collection information operation about the users connected to the network using the NetBIOS names and packet capturing technique. The information about the user will detect and analyze to accept the authorized users and refuse the unauthorized user [1].

After the basic terms have been presented we can introduce the proposed model in detail. Figure (1) shows the basic elements of the proposed system model and Figure (2) shows the system components:

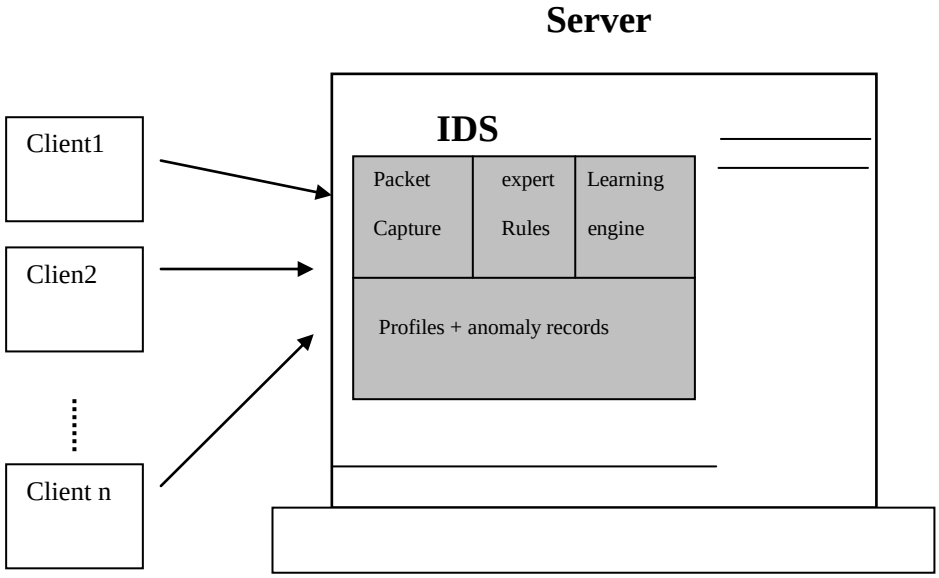


Figure (1) General Proposed System Structure

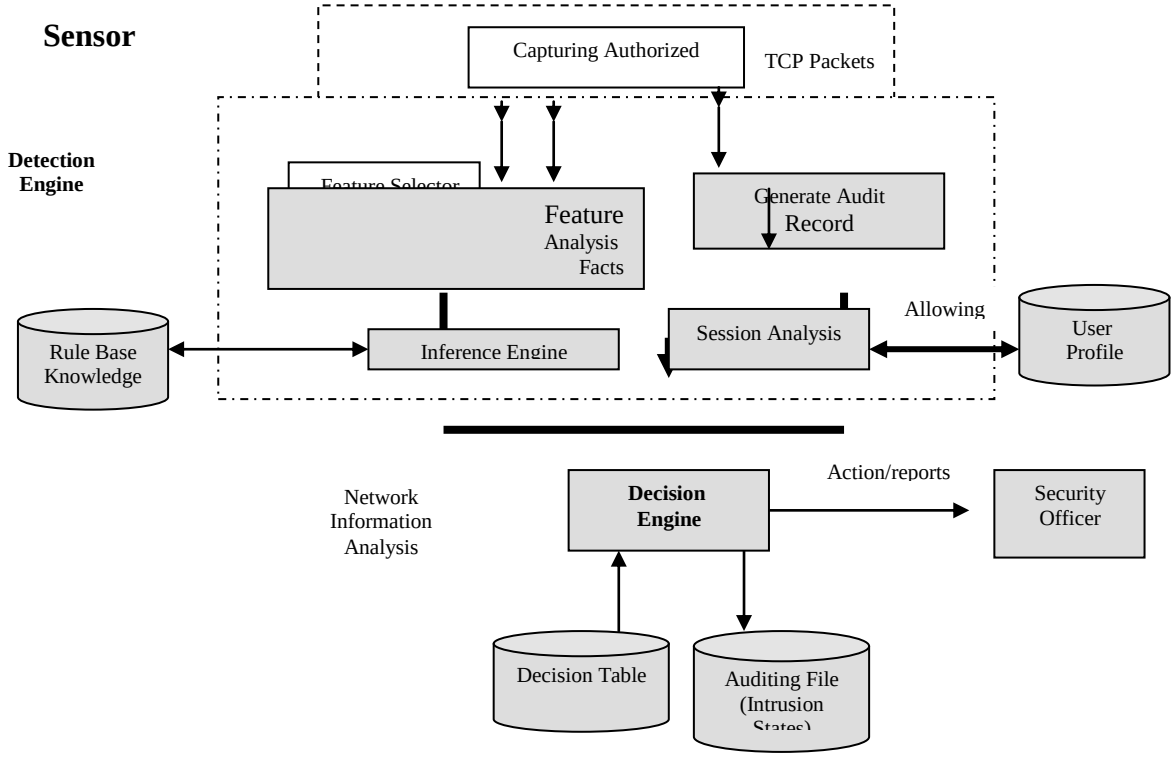


Figure (2) IDS Structure

2. Initialize the Profile of the Users and Learning phase:

The profile here is represented by two files which are the 'log files', the first one is initial log file which initialized at the beginning of the project by the administrator and could not be modified only on his permission.

The second file is the running files that to be constructed at the first session and then modified along each session. It contains the audit records

gained from the capture and analysis of the incoming packets. The profile consist of the following fields:
<User name, Password, Applications Vector, Time-start, Time-end, Day-work, Tricky-information>.
User name: the name is given to user
Password: authenticated field to user
Applications-vector: vector split into 6 fields refers to applications that the user has been authorized to access it.

Application 1 : represent the HTTP Service field.
Application 2: represent the Email Service field.
Application 3: represent the Server Messages field .
Application 4: represent the Chat services field .
Application 5: represent the FTP service . field

Time-start: It's Date-field (registered in The time that user starts his operations).
Time-end: It's Date-field (end time of user connect).
Tricky-questions: It's a text field contain information collected to personalize the subject like name of first school, name of grand father ..etc.

Input : Initialize status
Output: Detection users
Initialized the network connected.
Starts capturing and ping operations for detect connected users
Check the watermark in the packet if not found
Then apply the filter packet algorithm
Else goto step 4
Detect the users' names and passwords using the NetBIOS protocol.
Get the (IP, port, subnet mask, user name, password, time, no. of hours, personal information)
Collected the status for users
Apply the administration rules for each user
Save the status to records
End

To start connect user to the server, the user must enter the authentication information as (username and password) .The system will get the name and password of user and check to see if it is true or not .
Second function in the proposed system initializes the learning phase to update rules for any new events. Where, the proposed system will load the saved rules and the generation rule algorithm for starting analyzing and generation of the rules for the analyzed events.

3. Analyzing and Decision Making Stage

In this stage, the proposed system starting analyzes operation for the incoming packets to determine the events and operation for each user connect to the network. Also, the decision will make depending on the collected information about the users and administration rules Generally the basic flowchart of the proposed system that is as shown in Figure (3) .

3.1 Packet Capture : The first phase of the analyzing stage is the packet capture through which the network depends upon to monitor the traffic in order to reveal needed information, the values for each packet header fields packet capture the following field[3]:

H_length1: the IP header length, which is an integer value that defines the total length of the TCP header in four-byte words.

H_length2: the IP header length which is an integer value that defines the total length (header plus data) of the IP TCP n bytes.

Source_destination: it's an array of integer values represent the source and destination address .

S_port_D_port : it's an array of integer values represent the source port and destination port.

Flag1: represent SYN flag, which may be 0 or 1 depends on the value of flag in received packet.

Flag 2: represent ACK, which may be 0 or 1

Flag 3: represent FIN flag, which may be 0 or 1

Flag 4: represent PSH flag, which may be 0 or 1

Flag 5: represent URG flag, which may be 0 or 1

Seq1: pointer to the first byte of the data in packet segment.

The packet capture is concerned with capture the every packet that is originating from any node in the network that can be captured.

The flags field in the TCP header check to display the status of packet. The algorithm applied to accomplish the captured function.

3.2 **Packet Filtering** : The packet filter provides better access control mechanisms and either accept or reject packets passing through the network. the proposed system will apply the packet filter to prevent the unauthorized users from working on network. This algorithm based on packet information fields: Source IP address, Destination IP address, Protocol number, Source port number, and Destination port number and users profile[3].

3.3 Anomaly Detection (Session Detection)

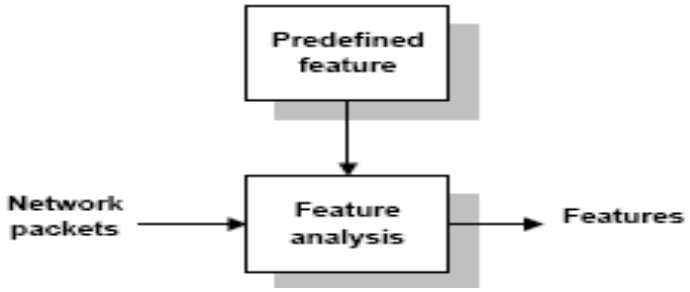
In this function, the system will try to recognize the session of the user does it. The session type can determine the application program that is needed to work for each user, there are a rule that determines the sets of authorized

application. The authorized application were recorded into the user record profile.

The user application can be determine from the session packets or NetBIOS packets. The proposed system will analyze the NetBIOS packet to determine the tasks and application of users.

3.4 Misuse Detection (Feature Extraction):

This type of detection involves extraction of the important parameters (features) and other information from network packets, the feature extraction will extract These selected features are used in Pre-Detector component [4]. The structure of FS is shown in Figure(3).



Figure(3) Feature Selector Component.

The features, in this algorithm, ICMP and TCP (SYN flooding attack, Address impersonation, Sequence number guession, session Hijacking, SYN Flood Attack, Ping of Death, Smurf Attack, Land Attack, and Teardrop Attack), are such as source IP, source port number, destination IP address, destination port number, TCP flags, window size, the interval value of two adjacent sequence numbers, fragment pointer, and interval time of two-consecutive packets. These features have been priory defined in the predefines feature component.

The predefined feature (K) is in the form of 6-tuples of parameters. Its characteristic can be defined system as follows.

K = (Flags, Ports, IPs, W, Fragment, ΔSEQ, ΔT)

Δ**SEQ** is an interval value of two sequence numbers.

Δ**T** is an interval time of two consecutive TCP segment.

The algorithm applied to accomplish the captured function is:

Input: sequence Packets.

Output: packet Information

Check the network traffic

: Copy the packets passing through network.

Analysis the packet (header (IP, TCP, and ICMP))

calculate the ΔSEQ and ΔT for the packet sequence.

Display results

Save results

End

4. Attack Detection

The proposed system was designed to determine the intrusion attack type. Attack detection operation is a preliminary analysis part of detecting phase. It analyzes events with a set of detection rule [5]. To identify intrusive patterns, by considering the features of K, sequence of packets must conform to several conditions. For example, if the SYN flooding attacks, a packet must have the SYN flag in its TCP header. For each packet we have the same destination IP address (IP) and the same destination port (Port). Finally, the packet must have the same window size (W). These conditions are described with the following rule:

START: IF flag is SYN THEN
IF IP are same THEN
IF Port are same THEN
IF W are same THEN
Goto Decision algorithm
ELSE goto START
ELSE goto START
ELSE goto START
ELSE goto START

The other attack of TCP also can be handling with same way. For example the Land attacks. this type of denial of service attack and depend on the TCP packet with a spoofed source address and port matching that of the destination. So, the rule of this type of attack is as shown:

START: IF flag is SYN THEN
IF Source IP = Destination IP THEN
IF Source Port= Destination Port THEN
IF no ICMP Packet type echo request is generated from the source IP address then smurf attack occurs
IF no more packets then
EXIT
END
END

4.1 The Proposed Rule-Based Learn Operation

The proposed learn operation consists of a rule translator, a library of runtime routines, and a set of collection routines. When using learn operation, rules and facts are written in the rule-based production rule specification format. The rule translator is then used to translate the specification into a C++ language rule-based system program(as update to the rules file).

In learning operation, the structure of a fact is specified by the administrator through a template definition referred to as a pattern type of rules. For example, to define a rule named *event* that consists of the four fields *event*

type (an integer), *return code* (an integer), *username* (a string), and *hostname* (a string) [5].

Facts from such a template definition could be constructed through the monitoring of audit records and asserted into the factbase for evaluation against the available production rules for example of a ptype declaration:

```
ptype[event event_type:int,
      return_code:int,
      username:string,
      hostname:string]
```

Fact evaluation is performed by the rule-based inference engine, where the attributes of the fact are mapped against the predicate expression(s) of each rule antecedent. For example, we may want to determine whether the asserted fact represents an unsuccessful login attempt, which we shall refer to form the statement in equation(1), S represents the set of all facts known to the rule-based fact base, and within which a production rule antecedent postulates the existence of a fact e that satisfies specific properties.

In the learning operation, the statement in equation (1) placed in the antecedent of the rule would be written as in Figure (4).

The term e :event allows one to assign an *alias* e to one fact (of possibly several) that satisfies the antecedent for the duration of the rule. The plus (+) sign after the opening bracket represents an existential quantifier that allows the rule to check for any fact that satisfies the conditions of the antecedent. Alternatively, a minus (-) sign searches for cases where no fact in the factbase satisfies the conditions of the antecedent. For example:

[-event|username == "GoodGuy"]

evaluates to true if there is no event in the factbase that has been asserted on behalf of "GoodGuy"

$$\left(\exists e\right)\Big((e \in S) \wedge \text{event}(e) \wedge (e_{event\_type} = login) \wedge (e_{return\_code} = bad\_password)\Big) \quad (1)$$

```
[+e:event|event_type == login,
          return_code == BAD_PASSWORD]
```

Figure (4) An example of fact matching

The plus and minus tests have corresponding assert and delete actions that can appear in the consequent of a rule, to assert a new fact of rule bad_login and give its fields initial values, we can write [+bad_login|username = e.username, hostname = e.hostname]

To be deleted from the factbase, a fact must be matched and given an alias in the antecedent before it can be deleted in the consequent. This is illustrated in the example of a complete rule named Bad Login in Figure (5).

```
1 rule[Bad_Login(#10;*):
2 [+e:event| event_type == login,
3 return_code == BAD_PASSWORD]
4 ==>
5 [+bad_login| username = e.username,
6 hostname = e.hostname]
7 [-|e]
8 [!|printf("Bad login for user %s from \
9 host %s\n", e.username, e.hostname)]
10 ]
```

Figure (5) An Example Of A Rule Declaration

The Bad Login rule in Figure (5) also demonstrates how the evaluation of an asserted fact can be used to derive subsequent facts that may themselves drive new inferences.

Using a mathematical notation, we can represent this state transition in rule factbase from S to a desired new state S0 as in equation (2).

$$\frac{(\exists e)((e \in S) \wedge event(e) \wedge (e_{event-type} = \log in) \wedge (e_{return-code} = bad\_password))}{+(S' = S - \{e\} \cup \{bad\_\log in(b) \mid (b_{username} = e_{username}) \wedge (b_{hostname} = e_{hostname})\})} \quad \ldots..(2)$$

Within parentheses after the rule name (line1), there is a semicolon-separated list of options. The option #10 means that this rule is given a ranking (priority) of 10. Priorities allow one to specify well-defined orders in the sequences for rule evaluation, and are primarily used for rules required to be evaluated first for initialization purposes, or that must be evaluated last to perform rules collection. The star option (*) indicates that the rule is repeatable, that is, the rule is allowed to fire repeatedly even if no other rule is fired in between. Thus, a key function of the consequent is to alter the state of the factbase such that the antecedent is not satisfied indefinitely (e.g., the consequent may mark or remove a fact). The arrow delimiter (==>) separates the antecedent and the consequent (line 4).

The [!|...] clause (line 8) within the consequent illustrates how the rule-based inference engine may call out to native C++ functions should action be warranted when the antecedent is evaluated to true. Both inference and action can be taken directly within the rule-based inference engine.

To further improve the performance of the rule-based system, rules can be disabled and enabled dynamically through actions in the consequents of rules. A rule can even disable itself, which means that it can fire once, at most, unless

enabled again by another rule. To disable a rule, we can put the following action in a consequent:
[-# rulename]

To enable a rule, we can change the minus sign in the above statement to a plus sign. In addition, a rule can be declared as disabled from start by adding a single minus sign to the list of options after the rule name, for example:
rule[rulename(#10;*;-):

Using these features, the preconditioned requirements that can enable or disable whole portions of the knowledge base can build, depending on the current state of the environment being monitored.

4.1.1 Rule Set for Detection of Failed Authentication Attempts

Table(1) Shows an example of rule set for detection failed authentication attempts [4].

Table(1) Rule set for detection failed authentication attempts.

rule[A1(*): [+e:bsm_eventˆA12] [?|e.header_event_type == ’AUE_login || e.header_event_type == ’AUE_telnet || e.header_event_type == ’AUE_rlogin || e.header_event_type == ’AUE_rshd || e.header_event_type == ’AUE_su] [?|e.return_return_value == ’INVALID_USER] [+cc: current_bl_cntr] [-max_bl_reached] ==> [+bad_login | timestamp = e.header_time, audit_seq_no = e.msequenceNumber, username = "invalid username", command = e.header_command, etype = e.header_event_type, hostname = e.subject_hostname, portID = e.subject_port_id, processID = e.subject_pid, textList = e.textList] [/cc| value += 1] [$|e:A12]]	rule[A2(*): [+e:bsm_eventˆA12] [?|e.header_event_type == ’AUE_login || e.header_event_type == ’AUE_telnet || e.header_event_type == ’AUE_rlogin || e.header_event_type == ’AUE_rshd || e.header_event_type == ’AUE_su] [?|e.return_return_value == ’INVALID_PWD] [+cc: current_bl_cntr] [-max_bl_reached] ==> [+bad_login | timestamp = e.header_time, audit_seq_no = e.msequenceNumber, username = e.subject_runame, command = e.header_command, etype = e.header_event_type, hostname = e.subject_hostname, portID = e.subject_port_id, processID = e.subject_pid, textList = e.textList] [/cc| value += 1] [$|e:A12]]
rule[A1(*): [+e:bsm_eventˆA12] [?|e.header_event_type == ’AUE_login ||	rule[A2(*): [+e:bsm_eventˆA12] [?|e.header_event_type == ’AUE_login ||

e.header_event_type == ’AUE_telnet || e.header_event_type == ’AUE_rlogin || e.header_event_type == ’AUE_rshd || e.header_event_type == ’AUE_su] [?|e.return_return_value == ’INVALID_USER] [+cc: current_bl_cntr] [-max_bl_reached] ==> [+bad_login | timestamp = e.header_time, audit_seq_no = e.msequenceNumber, username = "invalid username", command = e.header_command, etype = e.header_event_type, hostname = e.subject_hostname, portID = e.subject_port_id, processID = e.subject_pid, textList = e.textList] [/cc| value += 1] [$|e:A12]]	e.header_event_type == ’AUE_telnet || e.header_event_type == ’AUE_rlogin || e.header_event_type == ’AUE_rshd || e.header_event_type == ’AUE_su] [?|e.return_return_value == ’INVALID_PWD] [+cc: current_bl_cntr] [-max_bl_reached] ==> [+bad_login | timestamp = e.header_time, audit_seq_no = e.msequenceNumber, username = e.subject_runame, command = e.header_command, etype = e.header_event_type, hostname = e.subject_hostname, portID = e.subject_port_id, processID = e.subject_pid, textList = e.textList] [/cc| value += 1] [$|e:A12]]
rule[A3(*): [-max_bl_reached] [+cc:current_bl_cntr | value == ’x] [+ts:timeˆA3] ==> [!|printf("ALERT: Max Bad Logins \n")] [+max_bl_reached | value = 1] [$|ts:A3] [!|EXpertReport("Maha Intrusion Detection System", 1042, "description", ’pTypeString, "MAX LOGIN ALERT", "ruleName", ’pTypeString, "A3", "")]]	rule[A4(*): [+max_bl_reached] [+bc:bad_login] [+cc:current_bl_cntr] ==> [!|printf("(%s): %s from %s on %s port %d, \ PID = %d, time = %d, seq no = %d \n", bc.textlist, bc.command, bc.username, bc.hostname, bc.portID, bc.processID, bc.timestamp, bc.audit_seq_no)] [/cc|value -= 1] [-|bc]
rule[A5(*): [+mx:max_bl_reached] [-bad_login] ==> [-|mx]]	rule[A6(*): [+ts:timeˆA6] [-max_bl_reached] [+bc:bad_login] [+cc:current_bl_cntr] [?|(ts.sec - bc.timestamp) > ’y] ==> [/cc|value -=1] [-|bc] [$|ts:A6]]

4.2 Event Stream Format for SYN Flood Attack :

The SYN flood attack is a denial-of-service attack that prevents the target machine from accepting new connections to a given IP port [1].

The requirements for detecting the occurrence of a SYN flooding attack against a host are rather minimal. From the perspective of TCP/IP traffic monitoring, the analysis engine need only monitor SYNACK and ACK packet exchanges to identify incomplete TCP/IP handshakes.

In this example, the traffic monitor is placed on a segment of the network capable of observing traffic to and from the analysis target (the host being monitored). All SYN-ACK packets sent from—and ACK packets sent to—the analysis target are recorded, and the following event record is derived [6]:

Connection Event Format: <Event Type> <Timestamp> <Seq ID> <Client ID>

The Event Type field is simply a binary flag, which indicates whether the packet has its SYN and ACK flags enabled (which we can denote with 0), or only the ACK flag enabled (denoted by 1). The timestamp is a numeric encoding of the time at which the packet is observed from the monitor. The sequence ID represents the TCP Sequence ID field, which is used to associate client requests with server replies. Last, the Client ID can be used to identify the client who initiated the connection.

4.2.1 Rule-Based Fact Type Definitions

Table(2) illustrates the rule-based definitions of three example facts that are specified for use in performing the TCP SYN flooding analysis. The first rule-based, conn event, is used to assert the connection event .A connection events are captured by the network monitor, their fields can be mapped (one to one) to the fields of the conn event type, and the conn event type is then asserted into the factbase of the SYN flood intrusion detection system.

The open conn type is used to construct facts regarding half-open connections that are pending completion of the TCP/IP handshake. Note, although we use the shorthand name open conn, the fact actually represents the assertion that a TCP half-opened connection has been observed. The fields of the open conn contain the TCP sequence ID of the pending connection, a client ID string, the timestamp as copied from the connection event, and an expired flag used for rules collection by the production rules. Last, the bad connection fact, bad conn, maintains a running count of the number of bad connection requests detected through the observations of SYN-ACK and ACK packages between the analysis target and external clients.

4.2 The Rules For SYN Flood Detection.

The following illustrates one inference strategy that rule-based intrusion detection system can employ for deducing a TCP SYN flooding attack, using the fact definitions defined above. In addition, a few constants are referenced from the rule set, and are defined as follows [7]:

a. max bad conns: Number of bad connections tolerated before SYN flood alert.

b. expire time: Amount of time to wait on ACK before a connection is declared a bad connection.

c. bad conn life: Number of seconds that a bad connection fact will live before being released. The rules attempt to identify half-open TCP connections that expire beyond a user-defined waiting period. As we assert half-open connection facts into the factbase, we must include logic to recognize both when the connections are successfully completed and when half-open connection expire beyond the user-defined waiting period, from which we deduce the occurrence of a bad connection.

SYN flood attacks will result in excessive bursts of bad connections, which we monitor with rules that maintain a running count of bad connections over a sliding window of time. When the number of bad connections exceeds the maximum number for bad connections within the sliding time window, we raise an alert to denote the burst of no completed connection requests. The following is the rule set shown in Table(3).

d. create open conn: determines whether the event connection represents a SYN-ACK packet (from the monitor target), which records the TCP sequence number, the timestamp at which this half-opened connection was first observed, an expired flag to indicate when the half-open connection exceeds a time threshold, and the client ID.

e. destroy open conn: removes an open connection fact when the corresponding ACK packet is received from the client.

f. ignore spurious acks: removes events involving ACK packets that are not associated with a specific SYN-ACK pending connection. In practice, such packets are normal.

g. first bad conn: This and the following rule manage a running count of the set of bad connections observed by the inference engine. They are driven by time facts which are used to monitor whether there exists a half-open connection that has exceeded the expire time limit.

h. add to bad cons: is applied while the total number of bad conn facts is less than the maximum tolerated.

i. max open cons: is applied when the maximum number of bad conn facts is encountered during a burst of bad conn life time units.

j. free bad open cons: limits the amount of time that a bad open connection is counted against the system.

Table(2). Facts for TCP SYN flood detection.

type[bad_conn count:integer]	type[open_conn expired: integer, sec:integer, seq_id:integer, client_ID:string]	type[conn_event e_type:integer, sec:integer, seq_id:integer, client_ID:string]

Finally we must noted that, the final inference decision is attack type with attack ratio for increasing the correctness and the efficiency of the proposed system.

Table (3) Rule Set for Detection of TCP SYN Flood Attacks.

rule[add_to_bad_cons(*): [+ts:time] [+oc:open_conn|expired == 0] [?|(ts.sec -oc.sec) > 'expire_time] [+bc:bad_conn|count < 'max_bad_conns] ==> [/bc|count += 1] [/oc|expired = 1]]	rule[create_open_conn(*): [+ev:conn_event|e_type == 0] ==> [+open_conn |seq_id = ev.seq_id, sec = ev.sec, expired = 0, client_ID = ev.client_ID] [-|ev]]
rule[max_open_cons(*): [+ts:time] [+oc:open_conn|expired == 0] [?|(ts.sec -oc.sec) > 'expire_time] [+bc:bad_conn|count == 'max_bad_conns] ==> [!|syn_alert("SYN Attack: Last Host %s.\ SeqID = %d. Time = %d", oc.client_ID, oc.seq_id, oc.sec)] [/bc|count = 1] [/oc|expired = 1]]	rule[destroy_open_conn(*): [+ev:conn_event|e_type == 1] [+oc:open_conn|seq_id == (ev.seq_id -1)] ==> [-|oc] [-|ev]]
	rule[ignore_spurious_acks(*): [+ev:conn_event|e_type == 1] [-oc:open_conn|seq_id == (ev.seq_id -1)] ==> [-|ev]]
rule[free_bad_open_cons(*): [+ts:time] [+bc:bad_conn] [+oc:open_conn|expired == 1] [?|(ts.sec -oc.sec) > 'bad_conn_life] ==> [-|oc] [/bc|count -= 1]]	rule[first_bad_conn(*): [+ts:time] [-bad_conn] [+oc:open_conn|expired == 0] [?|(ts.sec -oc.sec) > 'expire_time] ==> [+bad_conn|count = 1] [/oc| expired = 1]]

5. Decision Engine:

There are two significant features, session permission, and inference decision. These two values could not be exactly specified with any well–form formula. for example, if the inference result is 60% SNY Flooding attack, while the session analysis result is allow copy session. These results lead to conflict in the final decision. The decision rule-based system starts with the threshold values. We assign membership values to inference result.. Membership function defines values into 3 levels, namely, low (L), medium (M), and high (H) as shown in Table (4).

Additionally, the variable R is the result of detection (ratio of attacking and the name of attack) . We create a set of rules in an IF-THEN form. These rules are derived from our experiments in detection of TCP/IP attacks and the surveying

of many hacking reports. The rules are described as follows(where IFR inference result, and SA =session result with (A = allow, NA=Not allow)).
Rule1: IF IFR =L AND SA= A THEN R=L
Rule2: IF IFR =L AND SA =NA THEN R=M
Rule3: IF IFR =M AND SA =A THEN R=M
Rule4: IF IFR =M AND SA =NA THEN R=H
Rule5: IF IFR =H AND SA =A THEN R=H
Rule6: IF IFR =H AND SA =NA THEN R=H

Value	High	Medium	Low
IFR	0.9	0.6	0.3

Table (4) The result of testing experiment.

After give the final decision, the administration will take his policies for prevent the attacker, while the proposed system will call the filter packet to stopping the attacker if the attacking ratio is high. Table (5) Shows the detection result of the proposed system of some attack types(each attack with attack ratio).

Attack type	Number	Misses	Score
Smurf	8	0	100%
Teardrop	4	0	100%
Land	2	0	100%
Ping of Death	5	0	99%
IP Sweep	3	0	96%
Satan	2	0	94%
Port Sweep	3	0	96%
Saint	2	0	89%
nmap	4	0	78%
Neptune	7	0	70%
mscan	1	0	55%
Total	43	0	88%

Table (5) The Proposed System Output Ratio of some attack

6. Discussion

From the time complexity comparison between the proposed system and the Neural Network Intrusion Detection System (NNIDS) [7] ,the results are as shown in Table (6).

Attack Name	Detection Time of the proposed system (sec)	Detection Time of the Neural Network system
system		

Detection Time of the proposed system (sec)
Smurf
TearDrop
Land
Ping of Death
IP Sweep
Saint
Nmap
Nepune
Mscan

Table (6) Comparison between the proposed system and the NNIDS to determine the detect

Note that the average of packet size (form 64 to 440 byte), each session (four connections are opened),the server can receive two packets at any one time ,each one is of (size 64 byte/sec).The Table (6) reveals ,that the detection time obtained based on the proposed system is better compared with the NNIDS [7]. Thus ,suggesting that the proposed system is less complex than NNIDS [7] .

7. Conclusions

The following points are concluded from the proposed system.

1. The system provides a sound basis for developing powerful real-time intrusion detection capable of detecting a wide range of intrusion related to attempted break-ins, masquerading (successful break-ins), system penetration and other abuses by legitimate users.
2. The system able to detect various TCP attacks with a better time and correct reaction.
3. The modular characteristics of the architecture allow it to be easily extended, configured and modified, either by adding new components, or by replacing components when they need to be updated.

4. The packet filtering technique is a useful technique to stopping the intruder work but this technique will not effect when the intruder is an authorized user with a high permissions.

5. The ability of the changing or modify the controlling rules with the update of the intrusion technique increase the efficiency of the proposed system to detect and treatments the newer types of attacking techniques.
6. Finally, the watermark technique help administrator to detect the intruder in very short time.

8. Future works:

The following are some suggestion for future works:

1. Using the encryption methods to encrypt the packet content before sends form client to server.
2. Improve the authentication by using the secure authentication techniques.
3. Improve proposed system by using the cover channel to transfer the secret message through the network.

References:

[1] Escamilla T. "Intrusion detection: Network security beyond the Firewall", John Wiley and Sons, Inc., 1998.

[2] Mahdi S. "Statistical approaches for Intrusion Detection System" MSc. Thesis, Department of Computer Science and Information Systems of the University of Technology.2000

[3] P.E Procter " The Practical Intrusion Detection Handbook" Prentice-Hall, Inc, 2001.

[4] S. Kumar "Classification & Detection of Computer Intrusions " Ph.D. Thesis, Purdue University, August 1995.

[5] R .F . Erbachar & B. Augustan "Intrusion Detection Data: Collection and Analysis" Dep. of Computer Science, University at Albania-sunny, 2000.

[6] Khazal H. "A simulated IDS Using Packet Capture", Ph.D. Thesis, Computer Science Dep. of the University of Technology, Baghdad, 2004 [4] Helmer .G, Wang J., Vasant and Lesmiller " Intelligent Agents for Intrusion Detection"

[7] "Network-based ID Model for Detecting TCP SYN Flooding " Dep. of Computer Engineering , Kasetsart University Bangkok. 2002