A Formal Method for Attack Modeling and Detection

Seyit Ahmet Çamtepe and Bülent Yener
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: {camtes,yener}@cs.rpi.edu

TR-06-01

Abstract

This paper presents a formal methodology for attack modeling and detection for networks. Our approach has three phases. First, we extend the basic *attack tree* approach [1] to capture (i) the temporal dependencies between components, and (ii) the expiration of an attack. Second, using the *enhanced attack trees* (EAT) we build a *tree automaton* that accepts a sequence of actions from input stream if there is a traverse of an attack tree from leaves to the root node. Finally, we show how to construct an *enhanced parallel automaton* (EPA) that has each tree automaton as a subroutine and can process the input stream by considering multiple trees simultaneously. As a case study, we show how to represent the attacks in IEEE 802.11 and construct an EPA for it.

I. Introduction

As the paradigm shifts toward pervasive networking, security challenges get harder. One of the most important aspect of security is to detect and prevent attacks in realtime. There are several types of Intrusion Detection Systems (IDS) designed to detect attacks [2]. Network Intrusion Detection Systems (NIDS) are used to detect attacks against a number of networked systems within their particular network environment. Network Node Intrusion Detection Systems (NNIDS) are located on critical systems, such as database servers and backup servers. Host-Based Intrusion Detection Systems (HIDS) look for suspicious activity at system logs, critical system files, and other resources. Anomaly-based intrusion detection systems define *normal* activity for a user and then look for any deviation from that normal activity.

While useful to detect illegal actions, IDS falls short of complex attacks that may not violate any rule explicitly. An example of such attacks is the *insider attacks*. There are mainly two major problems that intrusion detection systems faces with. First of all, insider attacks are sequence of more than one action therefore must be carefully modeled. An attack can be divided into more specific actions that can be detected by various IDS systems. For example hijacking an 802.11 MAC session requires attacker to *impersonate Access Point (AP)*, send *MAC disassociate* message to the client and *impersonate client*. These actions can further be defined as the proper combination of other smaller actions. Ordering of these actions is also critical. Attacker has to *impersonate AP* before sending *MAC disassociate* message to the client. Timing is another factor, because some of the actions can be usable in very short period of time. It is also possible to find more than one way of performing certain attacks, and an attacker may be trying either one, a subset or all of the ways simultaneously. In short, there should be a proper structure to model individual attacks with these properties. Second, problem is to detect such attacks or attack patterns in a stream of network activity data collected by various IDS systems. There should be mechanisms to detect an attack with a level of certainty before it is completed.

In this paper, we propose a formal methodology for attack modeling and detection for networks. This methodology can successfully capture complex attacks such as insider attacks. Our approach has three phases. First, we extend the basic *attack tree* approach [1] to capture (i) the temporal dependencies between components, and (ii) the expiration of an attack. Second, using the *enhanced attack trees* (EAT) we build a *tree automaton*. In this step, we build upon a Nondeterministic Finite Tree Automaton (NFTA) [3] and extend to design a new automaton that accepts a sequence of actions from input stream if there is a traverse of an attack tree from leaves to the root node. Finally, we show how to construct an *enhanced parallel automaton* (EPA) that has each tree automaton as a subroutine and can process the input stream by considering multiple trees simultaneously. As a case study, we show how to represent the attacks. We implement simulation program to test and evaluate our methods.

This paper organized as follows. In Section II, we introduce the background information on attack trees and NFTA. In Section III, we present our main results. In Section IV, we demonstrate how to apply the proposed technique to IEEE 802.11. In Section V, we test and evaluate our solution by simulation.

A. Terms and Definitions

Following terms and abbreviations are used throughout the paper:

ABBREVIATIONS			
AT	Attack Tree	AND	($\wedge$) Logical AND operation
EAT	Enhanced Attack Tree	O-AND	($\overrightarrow{\wedge}$) Ordered AND
NFTA	Nondeterministic Finite Tree Automaton	OR	($\vee$) Logical OR operation
NFETA	Nondeterministic Finite Enhanced Tree Automaton	TTL	Time To Live, life time
EPA	Enhanced Parallel Automaton	PC	Percent Confidence

II. Basics

Attack tree is first used by Schneier [1] to provide a formal way of describing the security of a system. Schneier proposes to represent attacks against a system in a tree structure where a goal is the root node and different ways of achieving that goal are leaf nodes. There are always chances to miss an attack while forming such attack trees. But, attack trees grow incrementally by time and they capture knowledge in a reusable form. Convery *et al.*[4] use attack tree to analyze potential threats to and using BGP (Border Gateway Protocol) from the adversaries perspective. Tuglular *et al.*[5] use different approach and classify attacks in three dimensions: incidents, response, and consequences. These dimensions then branch into new nodes and those nodes branch into new nodes until it can no longer be classified. This approach characterizes all possible incidents similar to an attack tree structure given in [1]. Magklaras *et al.*[6] refer human factor as the reason of incidents. Their model classify people into three dimensions: system role, reason of misuse, and system consequences. All of these models are based on prediction of attacks. Phyo *et al.*[7] propose a detection oriented approach to classify insider misuse based on the level of the system at which they might be detected. The main idea is that different types of misuses appear at different layers of a system. They classify the attacks into three layers: (i) network-level misuses, (ii) system-level misuses, (iii) application and data-level misuses.

Attack trees are the formal methods of modelling the attacks. More specifically, attacks are represented in a tree structure where the root node is the main goal, intermediate nodes are the subgoals, and leaf nodes are the ways to reach to the subgoals and finally to reach the main goal in turn. Children of a node in the tree can be of types: AND and OR. To reach a goal, all of its AND children or at least one of its OR children must be accomplished. This is similar for all subgoals down to leaves of the tree. It is quite easy to construct attack trees. First, possible attack goals must be identified. Each attack goal becomes root of its own attack tree. Then construction continues by considering all possible attacks against the given goal. These attacks form the AND and OR children of the goal. Next, each of these attacks becomes a goal and their children are generated. This process recursively goes down to leaves. In such a tree structure, an attack scenario to reach a main goal is the subtree which includes root node and all its AND along with at least one of its OR children. Same selection is made for all selected children (subgoals) recursively down to leaves. These selections form subtree of the given attack tree. An attack tree is complete if it contains a subtree for all possible attacks to fulfill given main goal. It is possible to assign different attributes to nodes on the tree such as time-to-live (TTL), cost, etc. By using such attributes, it is possible to extract attacks with certain properties, i.e. such information may be very useful in defining possible and feasible threats and invest for countermeasures.

Tree automata [3] have been initially designed in the late 50's in the context of circuit verification and found itself other application areas later on. In its basic form, it processes an input tree in bottom up manner, starting from leaves, moves up to root.

Definition 1: A Nondeterministic Finite Tree Automaton (NFTA) [3] is a tuple $A = (Q, F, Q_f, \Delta)$ where:

- Q is a set of states,
- $Q_f \subseteq Q$ is a set of final states,
- F is a set of $n-ary$ symbols (such as constant symbol a representing a leaf node, unary symbol $f()$ representing a node with one child, binary symbol $g(,)$ representing a node with two children, etc.),
- Δ is a set of transition functions in the form of:

$$f(q_1(x_1), \ldots, q_n(x_n)) \to q(f(x_1, \ldots, x_n))$$

where $f \in F$, $q, q_1, \ldots, q_n \in Q$ and $x_1, \ldots, x_n$ are all variables which can take on values from symbol set F.

Input to a NFTA is a tree which is expressed by a sequence of $n-ary$ symbols of the input alphabet F. Automaton processes tree in bottom-up fashion from leaves to the root. When the root is processed, automaton accepts the input tree, if it has reached to one of the final states in Q_f.

Example 2: ([3]) Let $F = \{or(,), and(,), not(), 0, 1\}$ where $or(,)$ and $and(,)$ are binary symbols and $not()$ is a unary symbol. Consider the automaton $A = (Q, F, Q_f, \Delta)$ where $Q = \{q_0, q_1\}$, $Q_f = \{q_1\}$, and Δ is:

$$\begin{array}{rclcrclcrclcrcl}
0 & \to & q_0 & , & 1 & \to & q_1 & , & not(q_0) & \to & q_1 & , & not(q_1) & \to & q_0 \ , \\
and(q_0,q_0) & \to & q_0 & , & and(q_0,q_1) & \to & q_0 & , & and(q_1,q_0) & \to & q_0 & , & and(q_1,q_1) & \to & q_1 \ , \\
or(q_0,q_0) & \to & q_0 & , & or(q_0,q_1) & \to & q_1 & , & or(q_1,q_0) & \to & q_1 & , & or(q_1,q_1) & \to & q_1 \ .
\end{array}$$

Consider the tree $and(not(0), or(1, 0))$ as given in Figure 1, the automaton defined above accepts this tree because when binary $and(,)$ symbol at the root is processed, automaton reaches to final state q_1. Figure 1 shows each step of the execution starting from the leaves. This automaton actually accepts all true boolean equations over F.

In the next section, first we will show how to enhance *attack trees* [1] to obtain *Enhanced Attack Trees* (EAT). Next, we will enhance *tree automaton* [3] and show that for each *enhanced attack tree*, it is possible to design an *Enhanced Tree Automaton* (ETA) which tell us if this tree is coded within the

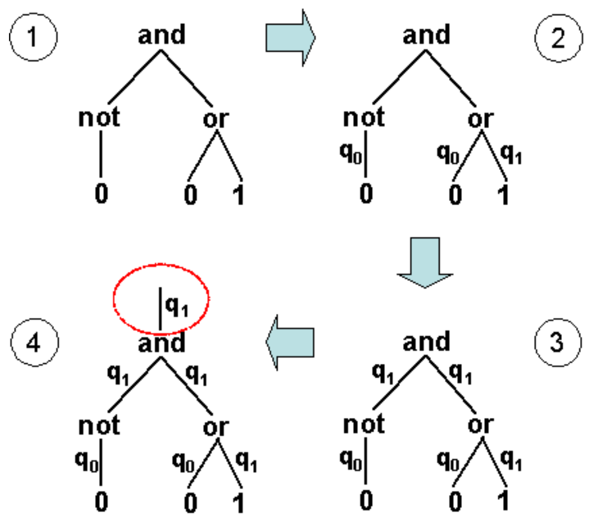

Fig. 1. Input tree $and(not(0), or(1,0))$ for the NFTA of Example 2. Step (1) shows the tree. During step (2), leaves are processed. Next in step (3), internal nodes $not(,)$ and $or(,)$ are processed. Finally in step (4), root is processed and final state is reached.

input stream. Finally, we will design a new *Enhanced Parallel Automaton* (EPA) which is union of the *enhanced tree automatons* and which can search more than one *enhanced attack tree* in parallel within the input stream.

III. Enhanced Parallel Automaton

A. Enhanced Attack Tree

In this work, we contribute several enhancements over *attack tree* [1] to increase its expressive power. First of all, there can be attack trees where goals or subgoals can be reached if all of their AND children are accomplished in the *given order* in time. It is not possible to use AND or OR type of children to enforce such an order. For example, *"Disconnecting a client"* in WLAN can be done by performing *"Eavesdrop MAC address of the AP"*, *"Impersonate AP"* and *"Send MAC disassociate message to the victim"* actions in this order. We call these types of children as type $O-AND$ (Ordered-AND) and denote with symbol $\overrightarrow{\wedge}$.

Our second contribution is time and confidence attributes of the nodes. *TTL* (Time-To-Live) attribute defines a life time for events at the leaf nodes and for subgoals at the interior nodes of the attack tree. In the example of *"Disconnecting a client"*, a client receiving a *"disassociate"* message disconnects from the AP but tries to reconnect soon again. Client restarts a well defined sequence of operations to reconnect. These operations includes probing the network to find the best AP, open mode or WEP authentication, and finally associating with the selected AP. Attacker has limited amount of time to finalize his attack before victim reconnects. Thus, if more than *TTL* time has elapsed since the event or the subgoal occurred, it must be expired. *TTL* attribute helps decrease number of $False^{+}$ which is one of the basic problems in intrusion detection systems.

PC (Percent Confidence) attribute defines chance of reaching to the goal (completing the attack) when a subgoal is accomplished. It is possible to report an attack with a confidence, before it is completed. To

Subgoals	Attack Scenarios	PC Attributes
B	$A\mathbf{(}B(C(D(a), b, c), d)\mathbf{)}$	$4/4 = 1$
C	$A(B(C\mathbf{(}D(a), b, c\mathbf{)}, d))$	$3/4 = 0.75$
D	$A(B(C(D\mathbf{(}a\mathbf{)}, b, c), d))$	$1/4 = 0.25$
E	$A(E\mathbf{(}F(e), f, g\mathbf{)})$	$3/3 = 1$
F	$A(E(F\mathbf{(}e\mathbf{)}, f, g))$	$1/3 = 0.33$

TABLE I
PC ATTRIBUTES FOR SUBGOALS IN ENHANCED ATTACK TREE OF FIGURE 2

be able to find *PC* attribute for a subgoal, all possible attack scenarios that includes the subgoal should be considered. A sperate *PC* for the subgoal, for each attack scenario is calculated and maximum among all is selected. *PC* is the ratio of all accomplished events until the subgoal over all events of the scenario. This attribute can be used to establish an early warning system and can help take precautions before possible damages of the attack.

Definition 3: An *Enhanced Attack Tree* (EAT) is an *attack tree* [1] where children of a node in the tree can be of types: AND, $O-AND$ and OR and where nodes has TTL and PC attributes.

Definition 4: An *attack path* starts from the goal of an enhanced attack tree and selects either one of the OR children or all of AND and $O-AND$ children recursively till reaching all the leaves. Each such *attack path* is also called as *attack scenario*.

Figure 2 presents a sample *enhanced attack tree* for *"Bypassing 802.1x"*. 802.1x authentication mechanism can be bypassed by either hijacking an authenticated session, or by playing MiM (Man-in-the-Middle) to steal credentials from a legitimate user [8]. Each event has *TTL* attribute, each subgoal has *PC* and *TTL* attributes, and edges are types of *AND*, *O-AND* and *OR*. Table I defines PC attributes for all subgoals. Root of the tree is the goal of the attack *"A - Bypassing 802.1x"*. To accomplish that goal, attacker has to perform either one of the subgoals *"B - Hijack 802.1x Authenticated Session"* or *"E - MiM 802.1x Session"*. Subgoal *"B - Hijack 802.1x Authenticated Session"* in turn can be accomplished by reaching subgoal *"C - Disconnect Client"* and *"d - Impersonate 802.1x Authenticated Client"* in this order (O-AND). Construction of the tree continues in this manner until all the leaves are all events. This tree has two possible *attack paths*, meaning two possible *attack scenarios* to reach the goal: (i) $A(B(C(D(a), b, c), d))$, (ii) $A(E(F(e), f, g))$.

Attack trees are generated so as to divide subgoals into as much detectable events as possible. But, existing network monitoring systems may be reporting events as well as subgoals used in the attack trees. When an occurrence of a subgoal is received while some of the children are not yet accomplished, we may consider that there are some other ways to realize that subgoal, therefore attack tree is not complete. Once an incomplete attack tree is detected, it can be reported to system administrator to redesign the corresponding attack tree. As an example, for the enhanced attack tree in Figure 2, it may be possible to receive a message indicating that a client is disconnected but event *"c - Send MAC Disassociate"* has not seen. In this case, we may suspect existence of some other mechanisms that can disconnect a client to hijack an 802.1x authenticated session.

As another example, consider the case that a user is passing through an 802.1x authentication process by using credentials of an existing active user. In this situation, we may conclude that attack tree is not complete because either an attacker hijacked a session and the legitimate user is trying to reconnect, or attacker has already recovered credentials of the legitimate user with a mechanism other than MiM type

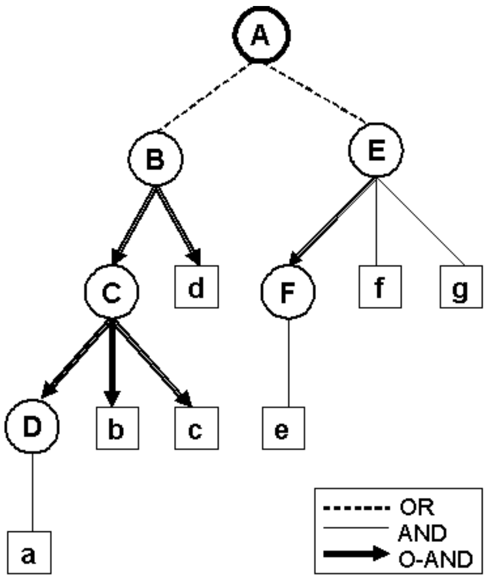

Fig. 2. Enhanced Attack Tree for *"Bypassing 802.1x"*. Capital letters [A .. Z] within circles are used to represent the goals and subgoals. Events at leaf nodes are represented with small letters [a .. z] within squares. Node descriptions are as follows: **A** - Bypass 802.1x, **B** - Hijack 802.1x Authenticated Session, **C** - Disconnect Client, **D** - Find 802.1x Authenticated Victim, **E** - MiM 802.1x Session, **F** - Find Unauthenticated Victim, **a** - Eavesdrop on 802.1x Authenticated Client, **b** - Use MAC Address of AP, **c** - Send MAC Disassociate, **d** - Impersonate 802.1x Authenticated Client, **e** - Eavesdrop on New Unauthenticated client, **f** - Impersonate AP, **g** - Impersonate New Unauthenticated Client.

of attack: guessing or dictionary attack for easy passwords, using spyware to steal passwords from the client computer, social engineering, breaking in configuration files of the system administrator, etc. Thus, attack trees provides not only a good model for the attacks, but also a self verifying system to check the completeness of the known attacks.

B. Enhanced Tree Automaton

There are several deficiencies of the *Nondeterministic Finite Tree Automaton* (NFTA) [3] therefore it is not suitable for use with enhanced attack trees. First of all, NFTA assumes a tree as the input, accepts it when the input is consumed and a final state is reached. In our system, input is a stream of messages within which we are looking for specific trees.

The most important issue with enhanced attack trees is that input stream may only provide events at the leaves of the tree. Since some of the subgoals may never appear in input stream, NFTA may never reach to a final state. Thus, we propose to use *derivation rules* as an enhancement to NFTA where a boolean variable is associated with each event.

Definition 5: A boolean variable *x* takes on value *true* if the corresponding event *x* appears within the input stream at most $TTL(x)$ time ago. It takes on value *false*, if it never appears, or if it appears more than $TTL(x)$ time ago.

Example 6: For the example in Figure 2, assume that *"C - Disconnect client"* is not in input stream but events *a*, *b* and *c* arrives. Instead of waiting for *"C - Disconnect client"* which may never appear, we may accept events *a*, *b* and *c* as boolean variables.

Goal or Subgoal	Boolean Expression
A	$(a\overrightarrow{\wedge}b\overrightarrow{\wedge}c\overrightarrow{\wedge}d) \vee (e\overrightarrow{\wedge}(f \wedge g))$
B	$a\overrightarrow{\wedge}b\overrightarrow{\wedge}c\overrightarrow{\wedge}d$
C	$a\overrightarrow{\wedge}b\overrightarrow{\wedge}c$
D	a
E	$e\overrightarrow{\wedge}(f \wedge g)$
F	e

TABLE II
ATTACK BOOLEAN EXPRESSIONS FOR THE THE GOAL AND SUBGOALS IN THE ENHANCED ATTACK TREE OF FIGURE 2

Thus, derivation rule for the example becomes $C = D\overrightarrow{\wedge}b\overrightarrow{\wedge}c = a\overrightarrow{\wedge}b\overrightarrow{\wedge}c$ where $\overrightarrow{\wedge}$ represents *O-AND* operation. We name this expression as *Attack Boolean Expression*. Boolean expression *C* evaluates to *true*, if all boolean variables are *true* and corresponding events *a*, *b* and *c* arrive in this order. If expression *C* evaluates to *true*, we assume the arrival of subgoal *C*. Subgoal *C* assumes both arrival time of event *a* (first arrival) and that of event *c* (last arrival). This is required because subgoal *C* itself may be child of type *O-AND* of another subgoal or the goal. Table II lists *attack boolean expressions* for the remaining subgoals and the goal in the enhanced attack tree of Figure 2.

In this work, we propose a new automaton technique and show how it can be used for detecting *enhanced attack trees* in a stream of messages. Our *enhanced tree automaton* design provides three basic functionality: (i) detecting attacks, (ii) detecting partial attacks and (iii) verifying completeness of the attack trees. Enhanced tree automaton is primarily based on the NFTA [3], but we improve it by introducing reporting states (also called partial attack states) in addition to final states (also called attack states), derivation rules for subgoals and backward transition rules to roll back the automaton when events and subgoals expire.

Definition 7: A Nondeterministic Finite Enhanced Tree Automaton (NFETA) is a tuple $A = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta$ where:

- Q is a set of states,
- $Q_{PA} \subseteq Q$ is a set of partial attack states,
- $Q_A \subseteq Q$ is a set of attack states,
- F is the input alphabet which consists of a set of $n - ary$ symbols,
- D is a set of derivation rules for the goal and subgoals in the form of boolean expressions of boolean variables representing events with AND, $O - AND$ and OR operations,
- Δ_F is a set of forward transition rules of the form:
$$f(q_1(x_1), \ldots, q_n(x_n)) \rightarrow q(f(x_1, \ldots, x_n)),$$
- Δ_B is a set of backward transition rules of the form:
$$q(f(x_1, \ldots, x_n)) \rightarrow q_1(x_1), \ldots, q_n(x_n)$$

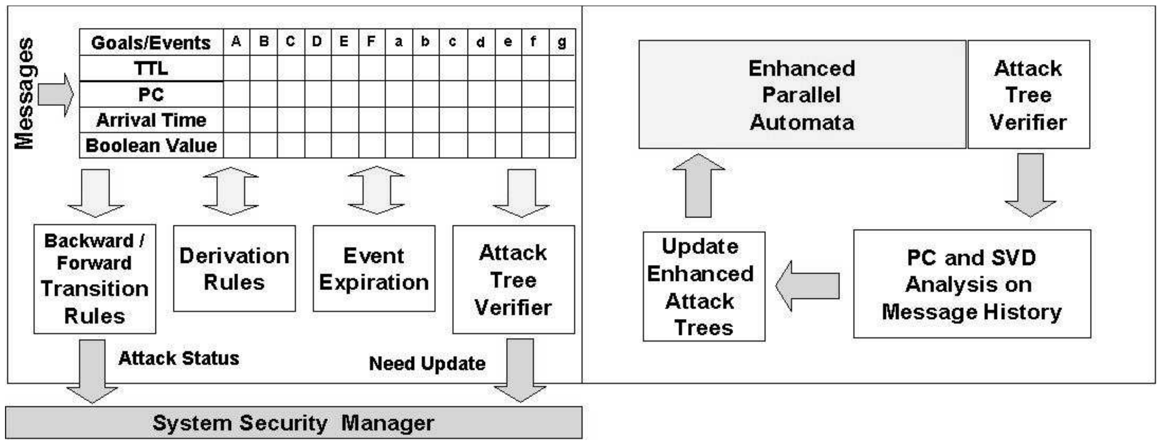

Fig. 3. Enhanced Tree Automaton System Design. States about the goals and events are stored in a table. When a message arrives, first time is updated, and events are checked against expiration. next, derivation, forward and backward rules are applied. Achieved goals and subgoals are reported. If an Attack tree is detected as being incomplete, it is reported for further analysis. An off line PC (Principle Component) or SVD (Singular Value Decomposition) analysis on history of messages is used to update incomplete attack trees.

where $f \in F$, $q, q_1, \ldots, q_n \in Q$ and $x_1, \ldots, x_n$ are all variables which can take any value from symbol set F. Input of this automaton is a stream of symbols of the input alphabet F. When automaton reaches to one of the partial attack states in Q_{PA}, it reports the attack with confidence PC. When automaton reaches to one of the attack states in Q_A, it reports attack along with the sequence of input events which caused this attack. Finally, automaton automatically resets to initial state by consuming the reported final state and continues its operation on input stream.

Similar to NFTA [3], NFETA uses an input alphabet F with n-ary symbols where a constant symbol (i.e. a, b, $\ldots$) represents a leaf node, an unary symbol $f()$ represents a goal or a subgoal with one child, a binary symbol $g(,)$ represents a goal or a subgoal with two children, etc.

Some of the subgoals in the enhanced attack trees may not appear in the input stream. But, we can use attack boolean expressions to evaluate subgoals and assume their existence. For this purpose NFETA includes *derivation rules* for each subgoal, similar to ones in Table II. When an event arrives, a boolean expression which includes that event is evaluated. If the boolean expression corresponding to the subgoal evaluates to true, we conclude that the subgoal is reached even though it is not in the input stream.

Forward transition rules are similar to transition rules of the NFTA [3]. When an event or a subgoal arrives, or when a subgoal is derived by derivation rules, corresponding symbol is used with forward transition rules to proceed on the tree and change the state.

Each event and subgoal has a life time which is defined with TTL attribute. If an event or a subgoal expires, derivation rules must be reevaluated to check whether any other subgoal expires. Next, for the expired subgoals and events, backward transition rules should be executed to roll back from current states. Backward transition rules are simply the reverse of forward transition rules. Figure 3 provides block diagram representation of the overall NFETA system design.

Following theorem shows construction of an NFETA for a given enhanced attack tree.

Theorem 8: There exist an NFETA for every enhanced attack tree.

Proof: (by construction) Given an enhanced attack tree, an NFETA with $A = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$

can be generated as follows:

- Q: Set of states can be obtained by assigning a separate state q_i for each edge in the tree along with an extra state for the goal.

- F: Set of symbols associated with the events at leaves along with $n-ary$ symbols associated with the goal and subgoals. Here n is the number of children of these subgoals and the goal.

- Q_{PA}: Partial attack states. Set of states which are associated with the edge connecting subgoals to its parent.

- Q_A: Attack state is the one associated with the goal at the root of the tree.

- D: Derivation rule for each subgoal is obtained starting from the leaves up to the root. Each event is associated with a boolean variable as defined in Definition 5 and each subgoal is expressed as a boolean expression of its children with operators AND, $O-AND$ and OR.

- Δ_F: For each event at leaf node x, add following forward transition rule where q_x is the state associated with the edge connecting node x to its parent:
$$x \rightarrow q_x.$$
For each subgoal X, add following forward transition rule where q_X is the state associated with the edge connecting subgoal to its parent, and $q_{x_1}, \ldots, q_{x_n}$ are the states associated with its children:
$$X(q_{x_1}, \ldots, q_{x_n}) \rightarrow q_X.$$

- Δ_B: backward transition rules are reverse of the forward transition rules:
$$q_X(X^{-}(q_{x_1}, \ldots, q_{x_n})) \rightarrow q_{x_1}, \ldots, q_{x_n}.$$

■

Example 9: For the example enhanced attack tree in Figure 2, we obtain following NFETA with $A^{802.1x} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$:

- $Q = \{q_a, q_b, q_c, q_d, q_e, q_f, q_g, q_A, q_B, q_C, q_D, q_E, q_F\}$
- $F = \{a, b, c, d, e, f, g, A(,), B(,), C(,,), D(), E(,,), F()\}$
- $Q_{PA} = \{q_B, q_C, q_D, q_E, q_F\}$ with PC attributes given in Table I
- $Q_A = \{q_A\}$
- D as provided in Table II
- Δ_F:

$$\begin{array}{ll} a \rightarrow q_a, & b \rightarrow q_b, \\ c \rightarrow q_c, & d \rightarrow q_d, \\ e \rightarrow q_e, & f \rightarrow q_f, \\ g \rightarrow q_g, & \\ A(q_B, q_E) \rightarrow q_A, & B(q_C, q_d) \rightarrow q_B, \\ C(q_D, q_b, q_c) \rightarrow q_C, & D(q_a) \rightarrow q_D, \\ E(q_F, q_f, q_g) \rightarrow q_E, & F(q_e) \rightarrow q_F. \end{array}$$

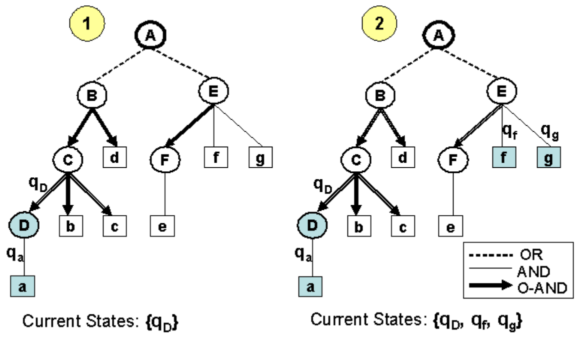

Fig. 4. NFETA of Example 9 on input a, f, f, g, b, c, e, d. In step (1), input symbol a is processed. Based on the transition rules, state q_a is reached. At the same time, derivation rules realize also that subgoal D has been reached, therefore transition rule advances state q_a to state q_D. At subgoal D, attack A is reported with $\%25$ confidence. In step (2), input symbols f, f and g are processed and states q_f and q_g are added among current states. Figures 5 and 6 present remaining steps.

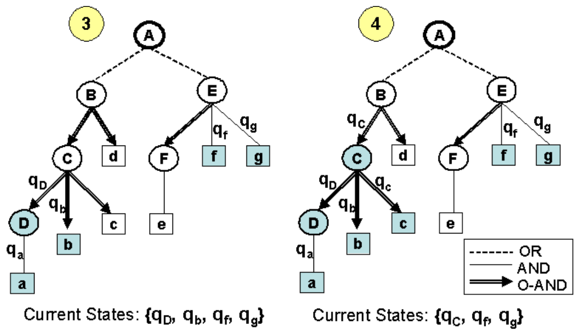
3
4
A
B
E
C
d
F
f
g
D
b
c
e
a
qD
qb
qa
qf
qg
qC
qc
OR
AND
O-AND
Current States: {qD, qb, qf, qg}
Current States: {qC, qf, qg}

Fig. 5. NFETA of Example 9 on input a, f, f, g, b, c, e, d. In step (3), input symbol b is processed and q_b is added among current states. In step (4), input symbol c is processed and q_c is added among current states. At the same time, derivation rules realize that subgoal C has been reached, therefore transition rule advances states q_D, qb and q_c to state q_C. At subgoal C, attack A is reported with $\%75$ confidence. Figure 6 presents remaining steps.

- Δ_B:
  $q_A(A^-(q_B, q_E)) \rightarrow q_B, q_E,$
  $q_B(B^-(q_C, q_d)) \rightarrow q_C, q_d,$
  $q_C(C^-(q_D, q_b, q_c)) \rightarrow q_D, q_b, q_c,$
  $q_D(D^-(q_a)) \rightarrow q_a,$
  $q_E(E(q_F, q_f, q_g)) \rightarrow q_F, q_f, q_g,$
  $q_F(F^-(q_e)) \rightarrow q_e.$

Figures 4, 5 and 6 presents how this NFETA operates an example input stream a, f, f, g, b, c, e, d.

C. Enhanced Parallel Automaton

Enhanced tree automaton is obtained from an enhanced attack tree, therefore it is designed to detect a single specific tree coded within stream of input data. More complex attacks which are combinations of the several enhanced attack trees can be handled in two ways. First we can design a separate enhanced tree automaton for each enhanced attack tree and feed a copy of the input stream to each independent

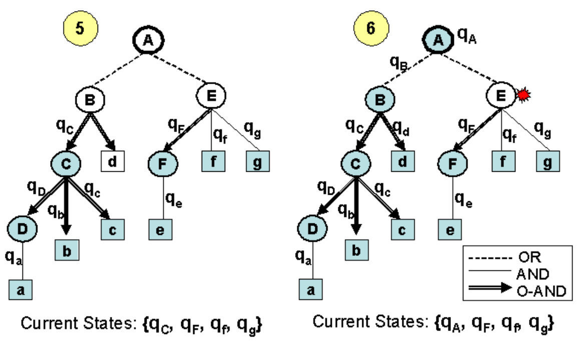
5
6
A
B
E
C
D
F
a
b
c
d
e
f
g
q_A
q_B
q_C
q_d
q_D
q_b
q_c
q_a
q_F
q_f
q_g
q_e
OR
AND
O-AND
Current States: {q_C, q_F, q_f, q_g}
Current States: {q_A, q_F, q_f, q_g}

Fig. 6. NFETA of Example of 9 on input a, f, f, g, b, c, e, d. In step (5), input symbol e is processed and q_e is added among current states. At the same time, derivation rules realize that subgoal F has been reached, therefore transition rule advances state q_e to state q_F. Attack A is not reported because $\%25$ confidence is not larger than last report with $\%75$. Since subgoal F is accomplished after arrival of symbols f and g, derivation rules fail to realize subgoal E. In step (6), input symbol d is processed and q_d is added among current states. Derivation rules first realize subgoal B and advances states q_C and q_d to state q_B. At subgoal B, attack A is reported with $\%100$ confidence. Next, derivation rules realize goal A and advances state q_B to q_A which is the final state.

automaton. Second, we can build an *Enhanced Parallel Automaton* (EPA) which is the combination of enhanced attack trees and uses single input stream to search several attacks in parallel.

Constructing an *Enhanced Parallel Automaton* for a set of n trees $T = \{T_1, T_2, \ldots, T_n\}$ is straightforward and consists of following two steps:

1) For each tree $T_i \in T$, generate its Nondeterministic Finite Enhanced Tree Automaton (NFETA) as defined in Theorem 8 with a common input alphabet F:

$$A^i = (Q^i, F, Q^i_{PA}, Q^i_A, D^i, \Delta^i_F, \Delta^i_B)$$

where $Q^1 \cap Q^2 \cap \ldots \cap Q^n = \emptyset$.

2) *Enhanced Parallel Automaton* is then defined as:

$$A = A^1 \cup A^2 \cup \ldots \cup A^n$$

where:

$$Q = Q^1 \cup Q^2 \cup \ldots \cup Q^n$$

$$Q_{PA} = Q^1_{PA} \cup Q^2_{PA} \cup \ldots \cup Q^n_{PA}$$

$$Q_A = Q^1_A \cup Q^2_A \cup \ldots \cup Q^n_A$$

$$D = D^1 \cup D^2 \cup \ldots \cup D^n$$

$$\Delta_F = \Delta^1_F \cup \Delta^2_F \cup \ldots \cup \Delta^n_F$$

$$\Delta_B = \Delta^1_B \cup \Delta^2_B \cup \ldots \cup \Delta^n_B$$

IV. Case Study: 802.11 Security

WLANs [9] are well accepted and brought a great flexibility to office and home networks. Mobility and portability provided by WLANs help attackers to better hide their malicious activity while providing them great access opportunities. The WLAN standards were developed without public review on security measures. This approach resulted in foundations of many different ways to crack WLAN security. Today, Internet is a good source of various efficient tools for eavesdropping wireless traffic, launching DoS types of attacks or cracking the encryption systems on use. These tools do not require any deep knowledge about the technology and can be easily used to launch wide range of active and passive wireless attacks.

Wireless Access Points (AP) provide physical and MAC layer security. Physical security consists of limiting RF signal availability within a particular perimeter by controlling direction and power of the antenna. There are five MAC layer security mechanisms: (i) hiding SSID (Service Set Identifier) information (SSID close mode of operation), (ii) MAC address based filters , (iii) WEP authentication and privacy mechanisms, (iv) 802.1x authentication mechanism, and (v) 802.11i privacy, integrity and key management mechanisms which are new.

In this case study, we first provide enhanced attack trees and enhanced tree automatons for attacks to bypass 802.11 security mechanisms: (i) SSID close mode of operation, (ii) MAC address based filters, (iii) WEP privacy, (iv) 802.1x authentication. Then we build an *Enhanced Parallel Automaton* which can detect these attacks in parallel.

SSID information is required to connect a WLAN. To provide security, SSID beacon messages sent by the Access Points (AP) can be disabled, so that only the client possessing the SSID information can probe and connect to the network. Attacker can learn SSID information of the network by listening client probe messages. Attacker can passively wait for a new client to arrive, or may disconnect a client forcing him to reconnect. Figure 7 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is as follows:

$$A^{SSID} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$$

where:

- $Q = \{q_h, q_b, q_i, q_j, q_G, q_H, q_I\}$.
- $F = \{h, b, i, j, G(), H(,,), I(,)\}$.
- $Q_{PA} = \{q_G, q_H\}$ with PC attributes 0.25 and 0.75 respectively.
- $Q_A = \{q_I\}$.
- D:
  $G = h$
  $H = h \overrightarrow{\wedge} b \overrightarrow{\wedge} i$
  $I = h \overrightarrow{\wedge} b \overrightarrow{\wedge} i \overrightarrow{\wedge} j$.
- Δ_F:
  $h \rightarrow q_h,$ $\quad b \rightarrow q_b,$
  $i \rightarrow q_i,$ $\quad j \rightarrow q_j,$
  $G(q_h) \rightarrow q_G,$ $\quad H(q_G, q_b, q_i) \rightarrow q_H,$
  $I(q_H, q_j) \rightarrow q_I.$
- Δ_B:
  $q_G(G^-(q_h)) \rightarrow q_h,$
  $q_H(H^-(q_G, q_b, q_i)) \rightarrow q_G, q_b, q_i,$
  $q_I(I^-(q_H, q_j)) \rightarrow q_H, q_j.$

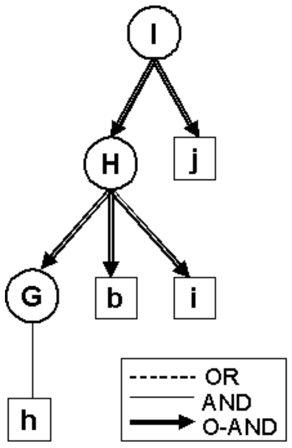

Fig. 7. Enhanced Attack Tree for *"Finding SSID in close mode"*. Capital letters [A .. Z] within circles are used to represent the goals and subgoals. Events at leaf nodes are represented with small letters [a .. z] within squares. Node descriptions are as follows: **G** - Find an active SSID victim, **H** - Disconnect SSID victim, **I** - Find SSID in close mode, **b** - Use MAC Address of AP, **h** - Eavesdrop on an active client, **i** - Send MAC disassociate to SSID victim, **j** - Eavesdrop on victims probe messages.

MAC filters are used to filter clients based on their MAC addresses. Attacker can passively monitor network for active clients and impersonate one of the clients as soon as it disconnects. Figure 8 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is as follows:

$$A^{MAC} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$$

where:

- $Q = \{q_k, q_l, q_J, q_K\}$.
- $F = \{k, l, J(), K(,)\}$.
- $Q_{PA} = \{q_J\}$ with PC attribute 0.5.
- $Q_A = \{q_K\}$.
- D:
  $J = k$
  $K = k \overrightarrow{\wedge} l$
- Δ_F:
  $k \rightarrow q_k, \qquad l \rightarrow q_l,$
  $J(q_k) \rightarrow q_J, \quad K(q_J, q_l) \rightarrow q_K.$
- Δ_B:
  $q_J(J^-(q_k)) \rightarrow q_k,$
  $q_K(K^-(q_J, q_l)) \rightarrow q_J, q_l,$

WEP privacy is basically provided by stream cipher $RC4$ with a 64 or 128 bit shared key and an Initialization Vector (IV). Shared secret is also used to authenticate clients where client simply encrypts and sends back the challenge provided by AP. WEP privacy is provided by xor operation of cleartext data with a key stream. Key stream is generated by $RC4$ algorithm using the shared secret and IV. WEP is secure as soon as key stream is not repeated. But, with 24 bit IV it is not rare to see key stream reuses. This problem of WEP is used to crack WEP key [10], [11]. An attacker can passively collect enough

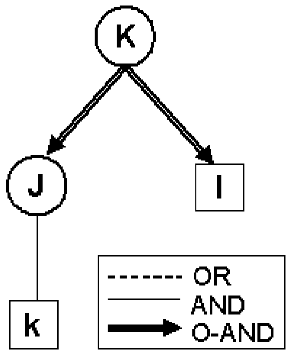

Fig. 8. Enhanced Attack Tree for *"Bypassing MAC filters"*. Capital letters [A .. Z] within circles are used to represent the goals and subgoals. Events at leaf nodes are represented with small letters [a .. z] within squares. Node descriptions are as follows: **J** - Find a MAC which is not blocked, **K** - Bypass MAC Filter, **k** - Eavesdrop on unblocked MAC, **l** - Impersonate unblocked client.

WEP traffic to use with WEP crack tools which are publicly available [12]. Figure 8 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is as follows:

$$A^{WEP} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$$

where:

- $Q = \{q_m, q_n, q_L, q_M, q_N\}$.
- $F = \{m, n, L(), M(,), N()\}$.
- $Q_{PA} = \{q_L, q_M\}$ with PC attributes 0.5 and 1 respectively.
- $Q_A = \{q_N\}$.
- D:
  $L = m$
  $M = m \overrightarrow{\wedge} n$
  $N = m \overrightarrow{\wedge} n.$
- Δ_F:
  $m \rightarrow q_m, \qquad n \rightarrow q_n,$
  $L(q_m) \rightarrow q_L, \qquad M(q_L, q_n) \rightarrow q_M,$
  $N(q_M) \rightarrow q_N.$
- Δ_B:
  $q_L(L^-(q_m)) \rightarrow q_m,$
  $q_M(M^-(q_L, q_n)) \rightarrow q_L, q_n,$
  $q_N(N^-(q_M)) \rightarrow q_M.$

802.1x provides authentication mechanism for 802.11. Absence of mutual authentication within 802.1x specification helps an attacker to play MiM (Man-in-the-Middle) attack, or to hijack session of an authenticated client. In MiM, attacker simply impersonates client to the AP and impersonates AP to the client. Since all the communication among client and AP is made through attacker, he can collect security credentials of the client. These credentials can be used by the attacker to pass 802.1x authentication. Figure 2 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is given in Example 9.

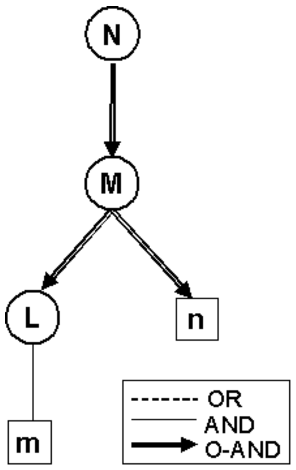

Fig. 9. Enhanced Attack Tree for *"Finding WEP key"*. Capital letters [A .. Z] within circles are used to represent the goals and subgoals. Events at leaf nodes are represented with small letters [a .. z] within squares. Node descriptions are as follows: **L** - Collect WLAN traffic, **M** - Cryptanalysis WEP key, **N** - Finding WEP key, **m** - Eavesdrop WLAN traffic, **n** - Crack WEP key.

Enhanced Parallel Automaton which detects 802.11 attacks can be constructed by union of individual enhanced tree automatons trees as defined in previous section:

$$A^{802.11} = A^{SSID} \cup A^{MAC} \cup A^{WEP} \cup A^{802.1x}$$

Figure 10 provides overall view of the Enhanced Parallel Automaton $A^{802.11}$ for attacks against 802.11.

V. Simulation and Results

We have implemented a simulation program to evaluate our enhanced parallel automaton technique. We used enhanced parallel automaton generated by the case study of 802.11 security. Simulation program has the structure similar to system design given in Figure 3. A perl program, *event generator* simulates network monitoring systems such as IDS (Intrusion Detection System) and generates random events. Both interarrival time and life time assignments for the events are roughly decided based on our observations and protocol expectations in 802.11. Basically, event set of *event generator* is divided into two parts. First part consists of *normal events*; events which are not included in any enhanced attack trees, and events which are in enhanced attack trees but can also be generated as a result of normal operations (i.e. *MAC Disassociate* messages). Second part consists of *attack events* which are remaining events of enhanced attack trees. Before generating an event, *event generator* makes a decision between sending a *random event* or an *attack sequence*. If *attack sequence* is selected, then one of the predefined *attack sequences* is sent. *Attack sequences* are attack paths which are generated from the enhanced attack trees of Figures 2, 7, 8 and 9 such as sequence (a, b, c, d) of Figure 2 which is enough conclude attack A and subattacks B, C, D. As another example sequence (D, b, c, d) is inserted to see attack tree verifier to create alarm since subgoal D arrives before its children, which means there may be another way of performing subgoal D and therefore attack tree is incomplete. If *random event* is selected, then a second selection is made between *normal* and *attack events*. Ratio of *attack events* in *random events* defines amount of noise in the log. Because, these events simulate the cases where there is no attack, but some of

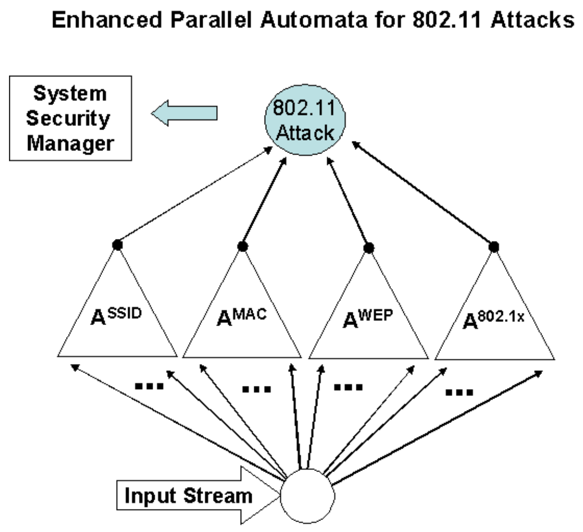

Fig. 10. Enhanced Parallel Automaton (EPA) for attacks against 802.11 security. EPA is $A^{802.11} = A^{SSID} \cup A^{MAC} \cup A^{WEP} \cup A^{802.1x}$.

the events accidentally appear in certain sequences and create a false alarm. Thus, ratio of *attack events* in *random events* is a simulation parameter.

While *event generator* generates a log file of given event number and given ratio of noise, it is processed by our simulation program. On receiving an event, simulation program updates its table where states about the goals and events are stored. Arrival updates the time and events are checked against expiration. Derivation rules, forward and backward transitions are performed. If goal or subgoal is reached, attack or subattack alarm is generated. If a goal or subgoal arrives before its children, that means our enhanced attack tree is incomplete thus an alarm is generated. Since attack sequences are predefined, expected alarms is compared against detected ones to decide on True or False alarms. In this work we are interested in amount of false positives which means an alarm is generated while there is no attack. False positives are due to noise in the input messages. Higher noise ratio means false positive. In term of security, false positives are preferred to false negatives which means alarm is not generated while there is an attack. But false positives creates false alarms and limits the efficiency of the network. False negatives requires human interaction and feedback since there is no evidence of attack in the system.

Figure 11 presents the result of the simulation runs. For each noise ratio (probability of *attack event*), we have calculated number of false positives for a log of size 10,000 events. We have repeated runs more than once and take average of the results. Simulation program successfully found the correct attacks, subattacks and incomplete attack tree cases for the attack sequences randomly planted into the log. Figure 11 shows false positive results of attack, subattack and incomplete tree alarms for probabilities $0.005, 0.01, 0.015, \ldots, 0.5$. As the noise increases in the log, number of false positives increases as expected.

A. Conclusion

In this paper we introduce a powerful technique to represent and detect attacks in a network. The contribution of this work is twofold. First it presents a new theoretical framework based on formal

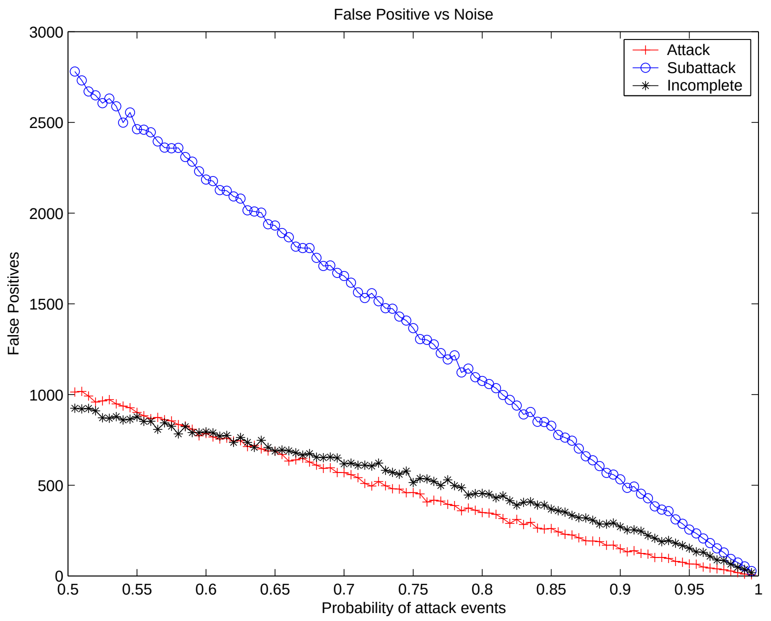

Fig. 11. Number of false positives for different noise levels over 10,000 events.

methods. Second, our technique advances the IDS approaches to capture more complex attacks such as insider attacks. We discuss how to realize the proposed technique in the context of IEEE 802.11 security. Simulation results show that it successfully detects attack sequences in large noisy input message stream.

References

[1] B. Schneier, "Attack trees: Modeling security threats," *Dr. Dobb's Journal*, December 1999.

[2] N. Einwechter. (2002, March) Preventing and detecting insider attacks using ids. SecurityFocus. [Online]. Available: http://www.securityfocus.com/infocus/1558

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, "Tree automata techniques and applications," April 2005, a book under construction. [Online]. Available: http://www.grappa.univ-lille3.fr/tata/

[4] S. Convery, D. Cook, and M. Franz, *BGP Attack Tree*, The Internet Engineering Task Force Working Draft Proposed Standard, 2001.

[5] T. Tuglular, "A preliminary structural approach to insider computer misuse incidents," in *1st European Anti-Malware Conference (EICAR 2000)*, 2000, Best paper.

[6] G. Magklaras and S. Furnell, "Insider threat prediction tool: Evaluating the probability of it misuse," *Computers and Security*, vol. 21, no. 1, December 2002.

[7] A. Phyo and S. Furnell, "A detection-oriented classification of insider it misuse," in *Third Security Conference*, April 2004.

[8] A. Mishra and W. A. Arbaugh, "An initial security analysis of the ieee 802.1x standard," University of Maryland, Tech. Rep. CS-TR-4328, 2002.

[9] *802.11*, IEEE 802.11 Standard, 2005. [Online]. Available: http://grouper.ieee.org/groups/802/11/

[10] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: the insecurity of 802.11," in *7th annual international conference on Mobile computing and networking*, 2001.

[11] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of rc4," in *8th Annual International Workshop on Selected Areas in Cryptography*, 2001.

[12] A. T. Rager. (2005) Wep crack and airsnort. SourceForge. [Online]. Available: http://wepcrack.sourceforge.net/