Modeling and Detection of Complex Attacks

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

***Abstract*—A *complex attack* is a sequence of temporally and spatially separated legal and illegal actions each of which can be detected by various IDS but as a whole they constitute a powerful attack. IDS fall short of detecting and modeling complex attacks therefore new methods are required. This paper presents a formal methodology for modeling and detection of complex attacks in three phases: (1) we extend basic *attack tree* (AT) approach to capture temporal dependencies between components and expiration of an attack, (2) using *enhanced AT* we build a *tree automaton* which accepts a sequence of actions from input message streams from various sources if there is a traversal of an AT from leaves to root, and (3) we show how to construct an *enhanced parallel automaton* that has each tree automaton as a subroutine. We use simulation to test our methods, and provide a case study of representing attacks in WLANs.**

I. Introduction

As the paradigm shifts toward pervasive networking, security challenges get harder. One of the most important aspects of network security is to detect and prevent attacks in realtime. There are several types of Intrusion Detection Systems (IDS) designed to detect attacks [1]. 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 and backup servers. Host-Based Intrusion Detection Systems (HIDS) look for suspicious activities at system logs, critical system files and other resources on a host. Signature-based IDS look for attributes of known malicious threats. Anomaly-based IDS define *normal* activities for a user and look for any deviation from them.

A complex attack is a multi-agent, multi-step and multi-stage attack. It is multi-agent because the attack may come from collaborating adversaries or an adversary with multiple identities. Adversary might be a legitimate user, as in *insider attacks*, or might be a legitimate or illegitimate user hiding behind innocent users as in *stealth attacks* [2]. It is multi-step because it is a sequence of temporally and spatially separated legal and illegal actions. Finally, it is multi-stage since each of these actions can be a complex attack itself. While useful to detect illegal actions, IDS fall short of complex attacks such as *insider* and *stealth* attacks which may not violate any rules explicitly. There are several characteristics of complex attacks which make IDS insufficient. First, a complex attack is a combination of legal and illegal actions each of which can be detected or prevented by various IDS but as a whole they constitute a powerful attack. For example, "*bypassing 802.1x authentication*" in WLAN requires an adversary to either "*hijack an 802.1x authenticated session*" or "*man-in-the-middle (MiM) an 802.1x session*". These actions can further be defined as proper combinations of other smaller actions. For "*hijack an 802.1x authenticated session*", an adversary has to first "*disconnect the client*", then "*impersonate the client*". "*Disconnecting the client*" in turn can be achieved first by "*using MAC address (impersonate) of the access point (AP)*" then by "*sending MAC disassociate message*" to the client. Second, the order of these actions is also critical. Adversary has to "*impersonate the AP*" before "*sending MAC disassociate message*" to the client. Thus, there exists partial or total order of the actions. Third, timing is another factor because some of the actions have a limited lifetime, and attacks which involve such actions must be completed within a time window. There is also more than one way of performing certain attacks, and an adversary may be trying a subset or all of them simultaneously. Thus, there is a need to model complex attacks with these properties. Efficient mechanisms are required to detect them in realtime within multiple streams of live network activity data collected by various IDS. Detection of completed attacks only helps us to understand them; it is required to detect attacks before they are completed. Furthermore, a partial attack may be reported to system administrators with the level of attack completed and the actions necessary to complete it.

In this paper, we propose a formal methodology for modeling and detection of complex attacks such as insider attacks. Our approach has three phases. First, we extend the basic *attack tree* approach [3] to capture the temporal dependencies between components and 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) [4] and extend it to design a new automaton that accepts a sequence of actions within an input stream if there is a traversal of an attack tree from leaves to root. Intermediate report states in the automaton are used to generate reports for partial attacks which include: (i) level of completion, (ii) sequences of actions required to complete the attack, (iii) sequence of contributor actions along with their arrival times, and (iv) source information of contributor actions where more than one identity means collaborating adversaries or an adversary with multiple identities. 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. Our approach does not consider interactions between hosts and network equipments. Only information about the hosts contributing to the attack and the victims is stored, therefore it is a scalable solution. As a case study, we show how to

This work was supported by Air Force Research Laboratory, Rome/NY.

represent and detect complex attacks in IEEE 802.11 WLANs. We use simulation to test and evaluate our methods. Proposed solution is implemented in perl for a testbed as our part of a WLAN security project supported by the Air Force Research Laboratory (Rome/NY). In this implementation, live messages from a variety of sensors are processed for detecting complex attacks in WLANs. Generated attack reports are used in dynamically adjusting users' access rights and privileges.

This paper is organized as follows. In Section II we briefly explain other approaches to attack modeling. In Section III we introduce background information on attack trees and nondeterministic finite tree automata. In Sections IV and V we present our main results. In Section VI we provide a case study which demonstrates application of proposed techniques to IEEE 802.11 WLANs. In Section VII we test and evaluate our solution by simulation. Finally, in Section VIII we conclude.

II. Related Work

Attack tree (AT) representation originates from fault trees which have been used in analyzing failing conditions of complex systems [5]. Attack trees are first used by Schneier [3] 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 the goal are leaf nodes. Convery *et al.* [6] use attack trees to analyze potential threats to and using Border Gateway Protocol (BGP) from adversaries' perspective. In [7] and [8], attack tree is enhanced by assigning context to attack nodes to provide a framework for modeling multi-stage network attacks. Attack trees can capture atomic steps of an attack where it is possible to assign cost and weight like parameters to these atomic steps. There is always a chance of missing an attack or an atomic step while forming such attack trees. But, attack trees grow incrementally by time and they capture knowledge in a reusable form. Attack trees have simple hierarchical structure where navigation in bottom-up or top-down manner is quite simple. It is also possible to process reusable branches or subtrees of an attack tree in a parallel fashion. There are some disadvantages of attack trees which should be addressed to improve their efficiency. Problem arises on preconditioning and timing. An attack may require its atomic actions happen in a strict time order. Moreover, some of the atomic actions can be valid during a certain period of time. Attack trees only have AND and OR types of children which restricts their expressive power.

Preconditioning is addressed in [9] and [5] by use of petri nets. A petri net consists of places, transitions and arcs connecting them. Places include tokens. A transition occurs when its preconditions are met, then tokens are moved from their input places to output places. *Attack Graph* (AG) is another attack modeling technique which is used in analyzing effects of local vulnerabilities on attacks [10], [11], [12] and [13]. In the basic scheme [10], scanning tools are used to determine vulnerabilities in a network. This vulnerability information is used along with host and connectivity information to generate attack graphs. Nodes in the graph are states of the network and arcs are the atomic attacks. Each path from start state to attack state is a series of exploits which leads to the attack. Attack graphs are mainly used to gather information about types of attacks the network is vulnerable to, and to make decisions such as the set of actions required to stop adversary. Finding minimal set of actions to stop adversary is shown to be an NP-complete problem [13]. Structure of attack graphs handles *AND* kind of relation among its atomic attacks by connecting them in serial and *OR* kind of relation by connecting them in parallel. Problem is to enforce an AND relation among the atomic attacks which can come in any order to reach the attack state. Such an unordered AND relation among n atomic attacks can be handled by parallel connection of all $n!$ permutations of these attacks. Another problem is that, attack graphs don't consider expiration of atomic attacks. When an atomic attack expires, attack state should be rolled back.

Attack languages are used to encode atomic actions of an attack into a suitable format [14]. In State Transition Analysis Technique (STAT) [15], attack languages are used to represent attack scenarios as a sequence of actions causing transition between security states of a computer system. In this work, we provide a link between attack languages and graph based attack representations. Namely, we enhance attack tree representation and propose a tree automaton technique to recognize attack trees within an input sequence of actions. Sequences of actions accepted by this automaton are actually words of an attack language.

In [16], a pattern matching approach is used to represent and detect intrusions. In [17] and [18], multi-stage attacks are detected based on their statistical characteristics. In [19], correlated attack modeling language is proposed to model multi-step attacks. A module is the basic unit with activity, pre-condition and post-condition sections, and modules are linked together to express a multi-step attack scenario. The purpose of this language is to provide an abstract attack model that can be shared among developers. In [20], attacks are classified in three dimensions: incidents, response and consequences. These dimensions then branch into new nodes and those nodes branch into new nodes until they can no longer be classified. This approach characterizes all possible incidents similar to an attack tree structure given in [3]. Magklaras *et al.* [21] refer to human factor as the reason of incidents. Their model classifies 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.* [22] 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 attacks into three layers: (i) network-level misuses, (ii) system-level misuses and (iii) application and data-level misuses.

III. Background

Attack trees are a formal method for modeling attacks. More specifically, an attack is 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 the main goal in turn. Root and intermediate

nodes can have AND and OR types of children. The goal is reached when all of its AND children or at least one of its OR children are accomplished. This is same 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 actions required for the given goal. These actions form AND and OR children of the goal. Next, each action becomes a goal and its children are generated. This process recursively goes down to the leaves. In such a tree structure, an attack scenario to reach the goal is a subtree which includes the root node and all of its AND along with at least one of its OR children. Same selection is made for all selected children (subgoals) recursively down to the leaves. These selections form subtree of the given attack tree. An attack tree is complete if it contains a subtree for all possible attack scenarios. It is possible to assign different attributes to the nodes such as time-to-live (TTL), cost, etc. By using such attributes, attacks with certain properties can be extracted which can be very useful in defining possible and feasible threats and in investing for countermeasures.

Tree automaton [4] has 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 up to the root.

Definition 1: A Nondeterministic Finite Tree Automaton (NFTA) [4] 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:

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

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 the input alphabet F.

Input to a NFTA is a tree which is expressed by a sequence of $n-ary$ symbols from 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 1: ([4]) 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:

Δ for $A = (Q, F, Q_f, \Delta)$					
0	$\rightarrow$	q_0	1	$\rightarrow$	q_1
$not(q_0)$	$\rightarrow$	q_1	$not(q_1)$	$\rightarrow$	q_0
$and(q_0, q_0)$	$\rightarrow$	q_0	$and(q_0, q_1)$	$\rightarrow$	q_0
$and(q_1, q_0)$	$\rightarrow$	q_0	$and(q_1, q_1)$	$\rightarrow$	q_1
$or(q_0, q_0)$	$\rightarrow$	q_0	$or(q_0, q_1)$	$\rightarrow$	q_1
$or(q_1, q_0)$	$\rightarrow$	q_1	$or(q_1, q_1)$	$\rightarrow$	q_1

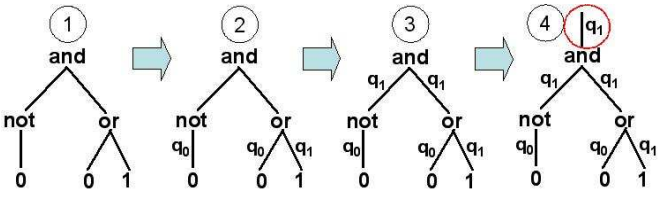

Fig. 1. Input tree $and(not(0), or(1, 0))$ for the NFTA of Example 1. Step (1) shows the tree. During step (2), leaves are processed. Next in step (3), internal nodes $not(,)$ and $or(,)$ are processed. Finally in step (4), root is processed and final state is reached.

Fig. 1 illustrates how $and(not(0), or(1, 0))$ is processed by automaton A. The automaton accepts this tree because when binary $and(,)$ symbol at the root is processed, automaton reaches to final state q_1. This automaton actually accepts all true boolean equations over F.

In the next section, first we will show how to enhance *attack trees* [3] to obtain *Enhanced Attack Trees* (EAT) which supports temporal dependencies between components and attack expiration. Then, we will enhance *tree automaton* [4] and show how to construct an *Enhanced Tree Automaton* (ETA) for a given enhanced attack tree. Finally, we will design a new *Enhanced Parallel Automaton* (EPA) which is union of the enhanced tree automata, and which can search more than one enhanced attack tree in parallel within input stream.

IV. Enhanced Parallel Automaton

A. Enhanced Attack Tree

In this work, we contribute several enhancements over *attack tree* [3] to increase its expressive power. First of all, there can be cases where goals or subgoals can be reachable only if all of their AND children are accomplished in a given time order. It is not possible to use AND ($\wedge$) or OR ($\vee$) 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 attack level attributes of the nodes. *TTL* (Time-To-Live) attribute defines a lifetime for actions at leaf nodes and for subgoals at 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 again soon. Client restarts a well defined sequence of operations to reconnect. These operations include probing the network to find the best AP, open mode or WEP authentication and finally associating with the selected AP. Adversary has limited amount of time to finalize his attack before the victim reconnects. Thus, if more than *TTL* time has elapsed since the action or the subgoal accomplished, it must be expired. *TTL* attribute helps decrease number of $False^+$ which is one of the basic problems in IDS.

Attack Level (AL) attribute defines amount or level of attack completed when a subgoal is accomplished. It is possible to report an attack (percent of the attack completed so far) before

Subgoal	Attack Scenarios	AL	Boolean Expr.
A	-	-	$(a \overrightarrow{\wedge} b \overrightarrow{\wedge} c \overrightarrow{\wedge} d) \vee (e \overrightarrow{\wedge} (f \wedge g))$
B	$A(B(C(D(a), b, c), d))$	1	$a \overrightarrow{\wedge} b \overrightarrow{\wedge} c \overrightarrow{\wedge} d$
C	$A(B(C(D(a), b, c), d))$	0.75	$a \overrightarrow{\wedge} b \overrightarrow{\wedge} c$
D	$A(B(C(D(a), b, c), d))$	0.25	a
E	$A(E(F(e), f, g))$	1	$e \overrightarrow{\wedge} (f \wedge g)$
F	$A(E(F(e), f, g))$	0.33	e

TABLE I
ATTACK LEVEL (AL) ATTRIBUTES AND ATTACK BOOLEAN EXPRESSIONS FOR GOALS/SUBGOALS IN ENHANCED ATTACK TREE OF FIG. 2

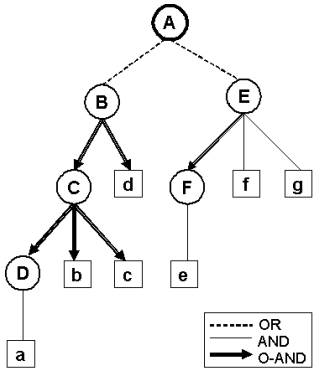

Fig. 2. Enhanced Attack Tree for "*Bypassing 802.1x*". Capital letters [A .. Z] within circles are used to represent the goals and subgoals. Actions at leaf nodes are represented with small letters [a .. z] within squares. Node descriptions are as follows: **A** - Bypass 802.1x, **B** - Hijack 802.1x 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.

it is completed. There can be different mechanisms to define AL attribute. In this work, we consider all possible attack scenarios which include the subgoal. A separate *AL* of the subgoal for each attack scenario is calculated and maximum among all is selected. *AL* is the ratio of all accomplished actions before the subgoal over all actions of the scenario. This attribute can be used to establish an early warning system and can help take precautions such as restricting access and user privileges before possible damages of the attack. In Section V we introduce *Attack Probability* (AP) attributes of subgoals and actions for probabilistic attack reporting where each action or subgoal may have different weight.

Definition 2: An *Enhanced Attack Tree* (EAT) is an attack tree [3] where children of a node in the tree can be of types: AND ($\wedge$), O-AND ($\overrightarrow{\wedge}$) and OR ($\vee$), and where each node has TTL and AL attributes.

Definition 3: An *attack path* starts from the goal and selects either one of OR children or all of AND and O-AND children recursively until reaching actions on the leaves. Each such attack path is also called as an *attack scenario*.

Fig. 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 Man-in-the-Middle (MiM) to steal credentials from a legitimate user [23]. Each action on the leaves has a *TTL* attribute, each subgoal has an *AL* and a *TTL* attribute, and edges are types of *AND*, *O-AND* and *OR*. Table I lists AL attributes for all subgoals. Root of the tree is the goal "*A - Bypassing 802.1x*". Adversary has to perform either one of the subgoals "*B - Hijack 802.1x Authenticated Session*" or "*E - MiM 802.1x Session*" to accomplish the goal. Subgoal "*B - Hijack 802.1x Authenticated Session*" in turn can be accomplished by reaching subgoal "*C - Disconnect Client*" and performing action "*d - Impersonate 802.1x Authenticated Client*" in this order (O-AND). Construction of the tree continues in this manner until all leaves are all actions. This tree has two possible *attack paths*, meaning two possible *attack scenarios* to reach the goal: (i) $A(B(C(D(a), b, c), d))$, and (ii) $A(E(F(e), f, g))$. Table I lists all attack scenarios involving each subgoal.

As another enhancement, we use an attack tree as a self verifying system. Attack trees are generated so as to divide subgoals into as much detectable actions as possible. But, existing network monitoring systems, sensors and IDS may be reporting actions as well as subgoals used in 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 the system administrator to redesign the corresponding attack tree. Thus, attack trees can grow incrementally by time. As an example, for the enhanced attack tree in Fig. 2, it may be possible to receive a message indicating that a client is disconnected but action "*c - Send MAC Disassociate*" has not been seen. In this case, we may suspect existence of some other mechanisms which can disconnect a client to hijack an 802.1x authenticated session.

As another example, consider the case where 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 adversary hijacked a session and the legitimate user is trying to reconnect or the adversary has already recovered credentials of the legitimate user with a mechanism other than MiM type of attack (i.e., guessing or dictionary attack for easy passwords, using spyware to steal passwords from the client computer, social engineering, breaking in configuration files). Thus, attack trees provide not only an effective model for attacks but also a self verifying system. Once the attack tree is detected to be incomplete, it needs to be updated by examining the message history from a variety of sensors and IDS. Detecting unknown attack scenarios within such a huge message history can be automated by using unsupervised learning and data analysis techniques such as PCA (Principle Component Analysis) and SVD (Singular Value Decomposition) [24]. These techniques can help us remove noise in message history to reach important actions. In [25] and [26], similar techniques are shown to be useful in extracting structure in large data sets.

B. Enhanced Tree Automaton

There are several deficiencies of the *Nondeterministic Finite Tree Automaton* (NFTA) [4] therefore it is not suitable for

use with enhanced attack trees. First of all, NFTA assumes a tree as input, accepts it when input is consumed and a final state is reached. In our system, input is a stream of messages coming from various sources such as IDS, sensors, firewalls and network packet analyzers. Specific trees are searched within such a live stream of messages.

The most important issue with enhanced attack trees is that input stream may only include actions on the leaves and some of the subgoals may never appear in input stream therefore NFTA may never reach to a final state. As an enhancement to NFTA, we propose to use *derivation rules* which require a boolean variable to be associated with each action.

Definition 4: A *boolean variable x* takes on value *true* if the corresponding *action(x)* appears within the input stream at most $TTL(x)$ time ago, *false* otherwise.

Example 2: In Fig. 2, assume that "*C - Disconnect client*" is not in input stream but actions *a*, *b* and *c* are. Instead of waiting for "*C - Disconnect client*" which may never appear, we may use derivation rule for $C = D\overrightarrow{\wedge}b\overrightarrow{\wedge}c = a\overrightarrow{\wedge}b\overrightarrow{\wedge}c$ 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 actions *a*, *b* and *c* arrive in this order. If expression *C* evaluates to *true*, we assume arrival of subgoal *C*. Subgoal *C* assumes both arrival time of action *a* (first arrival) and that of action *c* (last arrival). This is required because subgoal *C* itself may be a child of type *O-AND* of another subgoal or the goal. Table I lists *attack boolean expressions* for remaining subgoals and the goal in enhanced attack tree of Fig. 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 [4] but we improve it by introducing: (i) reporting states (called partial attack states) in addition to final states (called attack states), (ii) derivation rules for subgoals, and (iii) backward transition rules to roll back the automaton when actions and subgoals expire.

Definition 5: A Nondeterministic Finite Enhanced Tree Automaton (NFETA) is a tuple $A = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$ 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 is a set of $n-ary$ symbols,
- D is a set of derivation rules for the goal and subgoals in the form of boolean expressions,
- 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:
$$q(f(x_1, \ldots, x_n)) \rightarrow q_1(x_1), \ldots, q_n(x_n).$$

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 to this automaton is a stream of symbols from F. When automaton reaches one of the partial attack states in Q_{PA}, it reports the attack with level AL. When automaton reaches one of the attack states in Q_A, it reports the attack along with the sequence of input actions which caused this attack. We also report source information of input actions where more than one identity mean collaborating adversaries or an adversary with multiple identities.

Similar to NFTA [4], NFETA uses an input alphabet F with n-ary symbols where a constant symbol (i.e., a, b, ...) 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 enhanced attack trees may not appear in the input stream. But, we can use attack boolean expressions to evaluate subgoals and assume their arrival. For this purpose NFETA includes *derivation rules* for each subgoal as in Table I. When an action arrives, all boolean expressions which include the action is evaluated. If the boolean expression corresponding to a subgoal evaluates to true, we conclude arrival of the subgoal even though it may not be in the input stream.

Forward transition rules are similar to transition rules of NFTA [4]. When an action 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 to update the states.

Each action and subgoal has a lifetime which is defined with TTL attribute. If an action or a subgoal expires, derivation rules are reevaluated to check whether any other subgoal expires. Next, for the expired subgoals and actions, backward transition rules are executed to roll back from current states. Backward transition rules are simply the reverse of forward transition rules: given the set of current states and the symbol to be removed, the set of states before arrival of the symbol is reinstated. Fig. 5 step (5) provides a sample backward transition on action c.

Definition 6: Given an enhanced attack tree, an NFETA with A = (Q, F, Q_{PA}, Q_A, D, Δ_F, Δ_B) can be generated as follows:

- Q: Set of states are 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 are associated with the actions at leaves and $n-ary$ symbols with the goal and subgoals where n is the number of children.
- Q_{PA}: Partial attack states. Set of states which are associated with the edge connecting subgoals to their parents.
- Q_A: Attack state is the one associated with the goal at the root.
- D: Derivation rule for each subgoal is obtained starting from leaves up to the root. Each action is associated with a boolean variable as defined in Definition 4, and each subgoal is expressed as a boolean expression of its children with operators AND ($\wedge$), $O-AND$ ($\overrightarrow{\wedge}$) and OR ($\vee$).
- Δ_F: For each action at leaf node x, add $x \rightarrow q_x$ forward transition rule where q_x is the state associated with the edge connecting node x to its parent. For each subgoal X, add $X(q_{x_1}, \ldots, q_{x_n}) \rightarrow q_X$ forward transition rule

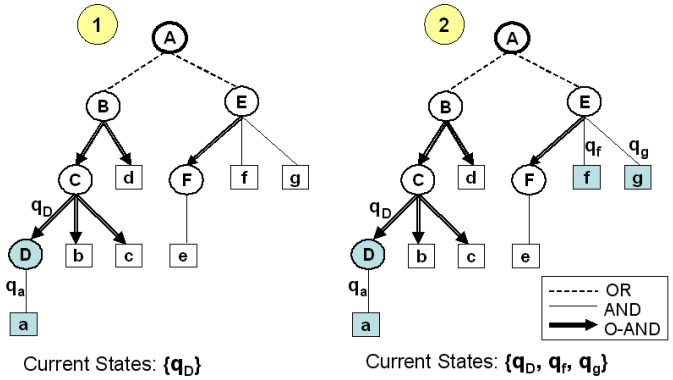

Fig. 3. NFETA of Example 3 on input $a, f, f, g, b, c, e, c, d$. In step (1), input symbol a is processed. Based on transition rules, state q_a is reached. At the same time derivation rules realize that subgoal D has been reached. Therefore transition rules advance state q_a to state q_D. At subgoal D, attack A is reported with level 25% . In step (2), input symbols f, f and g are processed and states q_f and q_g are added among current states. Fig. 4 presents remaining steps.

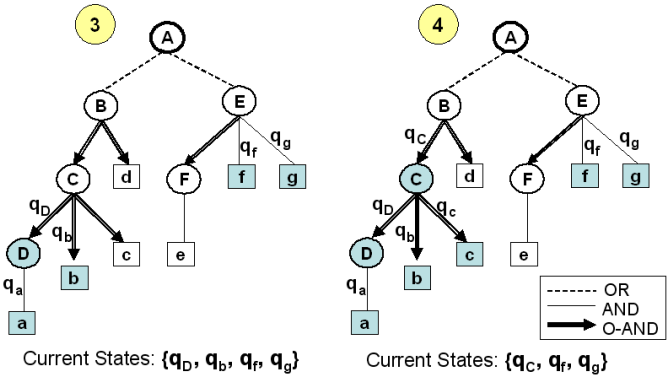

Fig. 4. NFETA of Example 3 on input $a, f, f, g, b, c, e, c, d$. In step (3), input symbol b is processed and q_b is added among current states. In step (4), input symbol c is processed and q_c is added among current states. At the same time derivation rules realize that subgoal C has been reached. Therefore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

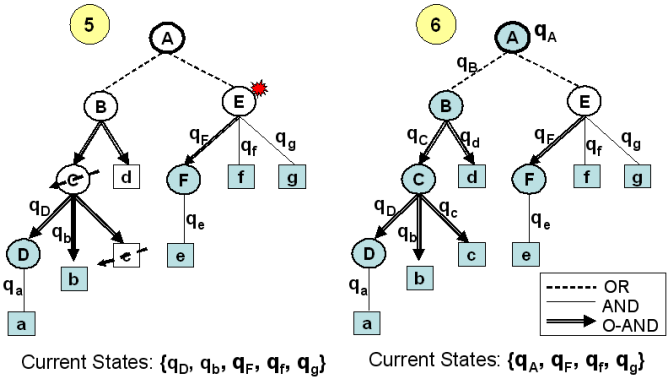

Fig. 5. NFETA of Example 3 on input $a, f, f, g, b, c, e, c, d$. In step (5), input symbol e is processed and q_e is added among current states. At the same time c expires and backward transition rules roll states q_c and q_C back to states q_D and qb. Derivation rules realize that subgoal F has been reached. Therefore transition rules advance state q_e to state q_F. Attack A is reported with level 25%. Since subgoal F is accomplished after arrival of symbols f and g, derivation rules fail to realize subgoal E. In step (6), input symbols c and d are processed and q_c and q_d are added among current states. Derivation rules first realize subgoal C and advance states q_D, qb and q_c to state q_C. Then realize subgoal B and advance states q_C and q_d to state q_B. At subgoal B, attack A is reported with level 100%. Next, derivation rules realize goal A and advance state q_B to q_A which is the final state.

where q_X is the state associated with the edge connecting the subgoal to its parent, and $q_{x_1}, \ldots, q_{x_n}$ are the states associated with its children.

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

Example 3: For the enhanced attack tree in Fig. 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 AL attributes as in Table I,
- $Q_A = \{q_A\}$,
- D: boolean expressions as provided in Table I,
- Δ_F:
  $a \rightarrow q_a$; $b \rightarrow q_b$; $c \rightarrow q_c$; $d \rightarrow q_d$;
  $e \rightarrow q_e$; $f \rightarrow q_f$; $g \rightarrow q_g$;
  $A(q_B, q_E) \rightarrow q_A$; $B(q_C, q_d) \rightarrow q_B$;
  $C(q_D, q_b, q_c) \rightarrow q_C$; $D(q_a) \rightarrow q_D$;
  $E(q_F, q_f, q_g) \rightarrow q_E$; $F(q_e) \rightarrow q_F$,
- Δ_B:
  $q_A(A^-(q_B, q_E)) \rightarrow q_B, q_E$;
  $q_B(B^-(q_C, q_d)) \rightarrow q_C, q_d$;
  $q_C(C^-(q_D, q_b, q_c)) \rightarrow q_D, q_b, q_c$;
  $q_D(D^-(q_a)) \rightarrow q_a$;
  $q_E(E(q_F, q_f, q_g)) \rightarrow q_F, q_f, q_g$;
  $q_F(F^-(q_e)) \rightarrow q_e$.

Fig. 3, 4 and 5 present how this NFETA operates on sample input stream $a, f, f, g, b, c, e, c, d$.

C. Enhanced Parallel Automaton

Enhanced tree automaton is obtained from an enhanced attack tree, therefore it is designed to detect a single 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 way is to design a separate enhanced tree automaton for each enhanced attack tree and feed a copy of the input stream to each independent automaton. Second way is to 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 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 in Definition 6 from a common input alphabet F as:
$$A^i = (Q^i, F, Q^i_{PA}, Q^i_A, D^i, \Delta^i_F, \Delta^i_B)$$
where $Q^1 \cap Q^2 \cap \ldots \cap Q^n = \emptyset$.
2) *Enhanced Parallel Automaton* is defined as:

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

where

$$Q = Q^1 \cup Q^2 \cup \ldots \cup Q^n,$$
$$Q_{PA} = Q^1_{PA} \cup Q^2_{PA} \cup \ldots \cup Q^n_{PA},$$
$$Q_A = Q^1_A \cup Q^2_A \cup \ldots \cup Q^n_A,$$
$$D = D^1 \cup D^2 \cup \ldots \cup D^n,$$
$$\Delta_F = \Delta^1_F \cup \Delta^2_F \cup \ldots \cup \Delta^n_F,$$
$$\Delta_B = \Delta^1_B \cup \Delta^2_B \cup \ldots \cup \Delta^n_B.$$

V. Probabilistic Attack Reporting

We assign AL (attack level) attribute to each subgoal in our enhanced attack trees. AL for a subgoal is the ratio of all accomplished actions of the subgoal to all actions of the attack scenario that involve the subgoal. There might be more than one such attack scenarios towards the attack goal thus a separate AL is calculated for each attack scenario and the maximum among all is selected. Once the subgoal is accomplished, the attack is reported with level AL. One problem here is that AL calculation for a subgoal considers each attack action with equal difficulty and weight. Consider the enhanced attack tree for "*A - bypassing 802.1x authentication*" in Fig. 2. Assume that the subgoal "*C - disconnect the client*" is accomplished, meaning that actions "*a - eavesdrop on 802.1x authenticated client*", "*b - use MAC address of AP*" and "*c - send MAC disassociate*" are seen. Then attack A is reported with an AL of 75% because only the action "*d - impersonate 802.1x authenticated client*" is missing out of four actions (a, b, c, d) which are sufficient for the attack A. But, performing action d is quite easier than performing action b, thus much more than 75% of the attack A is completed actually.

We introduce a new AP (attack probability) attribute for the actions and subgoals in enhanced attack trees. AP attributes describe the difficulty of performing actions or subgoals, and they are generated based on a simple statistical analysis on message history. Difficulty of an action or a subgoal is related to its frequency in message history, in other words, actions generated more often are assumed as easier to be accomplished. Let AP(x) be the AP attribute of an action x which is located on a leaf of an enhanced attack tree and let $\mathcal{H}$ be the message history, then AP(x) is the number of x in $\mathcal{H}$ divided by the total number of messages in $\mathcal{H}$. For a subgoal X, AP(X) = $MAX\{AP_{\mathcal{H}}(X), AP_G(X)\}$ where $AP_{\mathcal{H}}(X)$ is the number of X in $\mathcal{H}$ divided by the total number of messages in $\mathcal{H}$ and $AP_G(X)$ is generated from the AP attributes of children of subgoal X. Let $\wedge(X)$, $\overrightarrow{\wedge}(X)$ and $\vee(X)$ be AND, $O-AND$ and OR children of subgoal X respectively:

$$AP_G(X) = \prod_{x \in \wedge(X),\ x \in \overrightarrow{\wedge}(X)} AP(x)$$

$$AP_G(X) = \sum_{x \in \vee(X)} AP(x)$$

For the actions which can not be detected by any sensors, IDS or network analysis tools security administrators may set AP values manually. For example, AP value for the action "*a - eavesdrop on 802.1x authenticated client*" can be set as 1 because anybody can download and install a sniffer and passively eavesdrop the wireless traffic without being detected. Once the AP attributes for the actions and subgoals are set, we may provide the probability of an attack under the condition that a set of subgoals and actions are accomplished. Consider the enhanced attack tree of Fig. 2 and assume AP values for actions a, b, c and d be 1, 0.1, 0.5 and 0.2 respectively due to their easiness and frequency of appearance in the message history. Once actions a, b, c are seen, subgoal C is accomplished with $AP(C) = 1 \times 0.1 \times 0.5 = 0.05$. We may report attack A with conditional probability $Pr(A|C) = Pr(d|C) = AP(d)$ where $AP(d) = 0.2$ and action d is independent from actions of subgoal C. If we normalize the probabilities to 1 (i.e., $AP(d) \rightarrow 0.8$ and $AP(C) \rightarrow 0.2$), we may report attack A as being completed 80%.

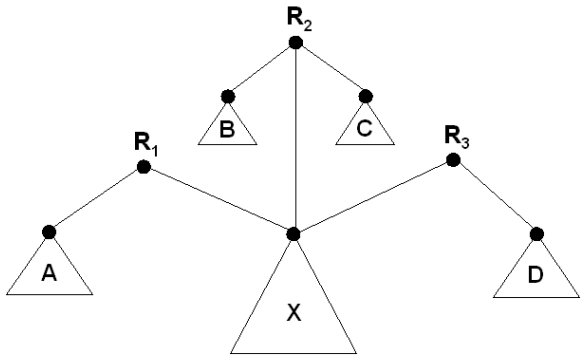

Fig. 6. Probabilistic attack reporting. Figure provides three enhanced attack trees for attacks R_1, R_2 and R_3. Subtree (a.k.a. subgoal) X is reused by these three trees. When subgoal X is accomplished, attacks R_1, R_2 and R_3 can be reported and ranked based on the probabilities $Pr(R_1|X) = Pr(A|X) = AP(A)$, $Pr(R_2|X) = Pr(B,C|X) = AP(B) \times AP(C)$ and $Pr(R_3|X) = Pr(D|X) = AP(D)$ if subtrees A, B, C, D and X are distinct and corresponding actions are independent.

It is possible that a subgoal (a.k.a. subtree) may be reused by more than one enhanced attack trees. In that case, when a subgoal is accomplished, all the attacks at the roots of these enhanced attack trees can be reported as partial attacks. AP (attack probability) attribute here can help us assign a probability for each; rank and report them to security administrators. Fig. 6 illustrates a sample scenario.

VI. Case Study: Modeling and Detection of Complex Attacks for 802.11 WLAN

WLANs [27] are well accepted and brought a great flexibility to office and home networks. Mobility and portability provided by WLANs help adversaries to better hide their malicious activity while providing them great access opportunities. 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 efficient tools for eavesdropping wireless traffic, launching DoS types of attacks or cracking the encryption systems on use. It doesn't require any deep knowledge about the technology to use these tools 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 sig-

nal 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.

IDS (e.g. attack graphs, attack languages, basic attack trees and pattern matching techniques) fail to detect complex attacks since they aren't capable of processing multiple facets of a complex attack (i.e., timing, temporal separation and ordering such as O-AND, AND and OR properties) concurrently. For example, consider insider attacks in 802.11 WLANs in which an attacker can "*hijack or MiM a session*", "*disconnect a client*", "*impersonate a client or an AP*", etc. Although some of these actions can be detected by IDS, they can't combine the pieces to obtain a macroscopic view which indicates a complex attack. In this case study, we first provide enhanced attack trees and enhanced tree automata for complex 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 beacon messages sent by Access Points (AP) can be disabled so that only a client possessing correct SSID information can probe and connect. An adversary can only obtain SSID information by listening client probe messages. Adversary can either wait for a new client to arrive, or can disconnect a client forcing him to reconnect. Fig. 7 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is $A^{SSID} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$ 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 AL 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: $h \rightarrow q_h$; $b \rightarrow q_b$; $i \rightarrow q_i$; $j \rightarrow q_j$; $G(q_h) \rightarrow q_G$; $H(q_G, q_b, q_i) \rightarrow q_H$; $I(q_H, q_j) \rightarrow q_I$,
- Δ_B: $q_G(G^-(q_h)) \rightarrow q_h$; $q_I(I^-(q_H, q_j)) \rightarrow q_H, q_j$; $q_H(H^-(q_G, q_b, q_i)) \rightarrow q_G, q_b, q_i$.

MAC filters are used to filter clients based on their MAC addresses. An adversary can passively monitor network for active clients and impersonate one as soon as it gets disconnected. Fig. 7 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is $A^{MAC} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$ where:

- $Q = \{q_k, q_l, q_J, q_K\}$,
- $F = \{k, l, J(), K(,)\}$,
- $Q_{PA} = \{q_J\}$ with AL attribute 0.5,
- $Q_A = \{q_K\}$,
- D: $J = k$; $K = k \overrightarrow{\wedge} l$,
- Δ_F: $k \rightarrow q_k$; $l \rightarrow q_l$; $J(q_k) \rightarrow q_J$; $K(q_J, q_l) \rightarrow q_K$,
- 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 the stream cipher $RC4$ with a 64 or 128 bit shared secret and an Initialization Vector (IV). Shared secret is also used to authenticate clients where clients simply encrypt and send 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 [28], [29]. An adversary can passively collect enough WEP traffic to use with WEP crack tools which are publicly available [30]. Fig. 7 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is $A^{WEP} = (Q, F, Q_{PA}, Q_A, D, \Delta_F, \Delta_B)$ 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 AL 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: $m \rightarrow q_m$; $n \rightarrow q_n$; $L(q_m) \rightarrow q_L$; $M(q_L, q_n) \rightarrow q_M$; $N(q_M) \rightarrow q_N$,
- 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 with EAP-MD5 (RFC 2284) helps an adversary to play Man-in-the-Middle (MiM) attack, or to hijack session of an authenticated client. In MiM, adversary simply impersonates the client to the AP and impersonates the AP to the client. Since all the communication among client and AP is made through adversary, adversary can collect security credentials of the client. These credentials can be used to bypass 802.1x authentication. Fig. 2 provides enhanced attack tree for the attack. Nondeterministic Finite Enhanced Tree Automaton (NFETA) for this tree is given in Example 3.

Enhanced Parallel Automaton which detects 802.11 attacks can be constructed by union of individual enhanced tree

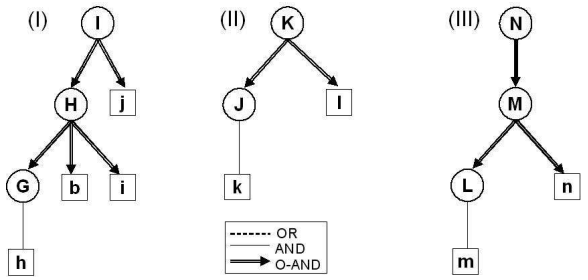

Fig. 7. Enhanced Attack Trees: (I) "*Finding SSID in close mode*", (II) "*Bypassing MAC filters*", and (III) "*Finding WEP key*". Capital letters [A .. Z] within circles are used to represent the goals and subgoals. Actions at leaf nodes are represented with small letters [a .. z] within squares. Node descriptions are as follows: **G** - Find an active SSID victim, **H** - Disconnect SSID victim, **I** - Find SSID in close mode, **J** - Find a MAC which is not blocked, **K** - Bypass MAC Filter, **L** - Collect WLAN traffic, **M** - Cryptanalysis WEP key, **N** - Finding WEP key, **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, **k** - Eavesdrop on unblocked MAC, **l** - Impersonate unblocked client, **m** - Eavesdrop WLAN traffic, **n** - Crack WEP key.

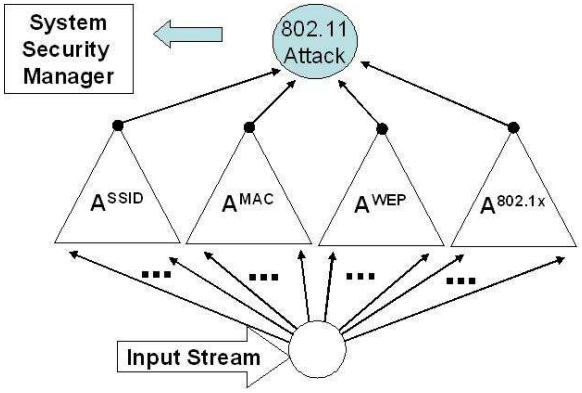

Fig. 8. Enhanced Parallel Automaton (EPA) for attacks against 802.11 security. EPA is $A^{802.11} = A^{SSID} \cup A^{MAC} \cup A^{WEP} \cup A^{802.1x}$.

automata as defined in Section IV-C:

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

Fig. 8 provides overall view of the Enhanced Parallel Automaton $A^{802.11}$ for complex attacks against 802.11 WLAN.

VII. Simulation and Results

We have implemented our enhanced parallel automaton technique. We use enhanced parallel automaton generated by the case study of complex attacks for 802.11 WLAN. Our implementation has a structure similar to system design given in Fig. 9 and 10.

A perl program, named as *action generator*, simulates detection and monitoring systems such as IDS and generates a random sequence of actions (input messages in Fig. 9). Both interarrival time and lifetime assignments for the actions are decided based on our observations and protocol expectations in 802.11 WLANs. Basically, action set of *action generator* is divided into two parts. First part consists of *normal actions*: actions which are not included in any enhanced attack trees, and actions which are in enhanced attack trees but can also be generated as a result of normal WLAN operations (i.e., "*MAC Disassociate*" message). Second part consists of *attack actions* which are the remaining actions in enhanced attack trees. Before generating a random action, *action generator* makes a decision between sending a *random action* or an *attack sequence*. If *attack sequence* is selected, then one of the predefined *attack sequences* is sent. *Attack sequences* are attack paths (Definition 3) which are generated from enhanced attack trees of Fig. 2 and 7 (i.e., the sequence (a, b, c, d) of Fig. 2 which is enough to conclude attack A and subattacks B, C, D). Other example sequences, such as (D, b, c, d), are inserted for 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 attack tree is incomplete. If *random action* is selected, then a second selection is made between *normal* and *attack actions*. Ratio of *attack actions* in *random actions* defines amount of noise in the log. Because, these actions simulate the cases where there is no attack but some of attack actions appear in input messages and create

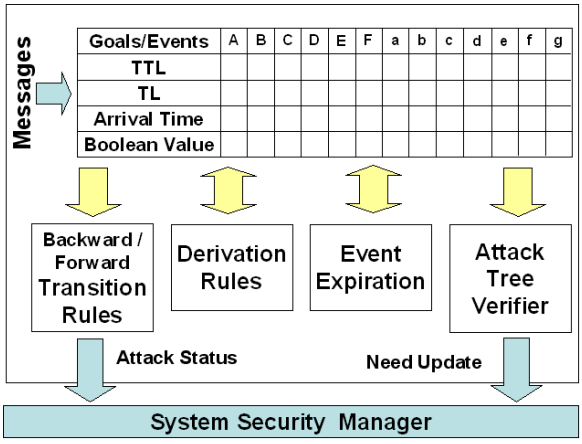

Fig. 9. Enhanced Tree Automaton System Design. States about the goals and actions are stored in a table. When a message arrives, time is updated, and actions are checked against expiration. Next, derivation, forward and backward transitions rules are applied. Achieved goals and subgoals are reported. If an attack tree is detected as being incomplete, it is reported for further analysis as in Fig. 10.

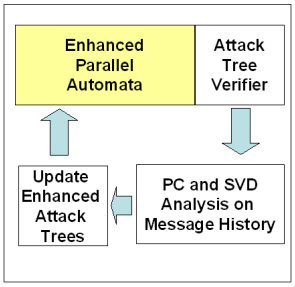

Fig. 10. Enhanced Tree Automaton System Design. If an attack tree is detected as being incomplete, it is reported for further analysis. An offline PCA (Principle Component Analysis) or SVD (Singular Value Decomposition) [24] on history of messages is used to update incomplete attack trees.

false alarms. Thus, ratio of *attack actions* to *random actions* is a simulation parameter.

While *action generator* generates a log file for a given number of actions and given ratio of noise, output sequence is processed by our *processing module* which is implemented in perl for a testbed as a part of WLAN security project supported by Air Force Research Laboratory (Rome/NY). On receiving an action, *processing module* updates its table where states about the goals and actions are stored. Arrival times are updated and actions are checked against expiration. Derivation, forward and backward transition rules are performed. If a goal or a subgoal is reached, attack or partial attack alarm is generated. If a goal or subgoal arrives before its children, meaning that our enhanced attack tree is incomplete, an "*attack tree is incomplete*" alarm is generated. Since attack sequences are predefined, expected alarms are compared against detected ones to decide on True and False alarms. In this work, we are interested in amount of $False^+$ which means an alarm is generated while there is no attack. $False^+$ are due to noise in input messages. Higher noise ratio means more $False^+$. In terms of security, $False^+$ are preferred to $False^-$ which mean an alarm is not generated while there is an attack.

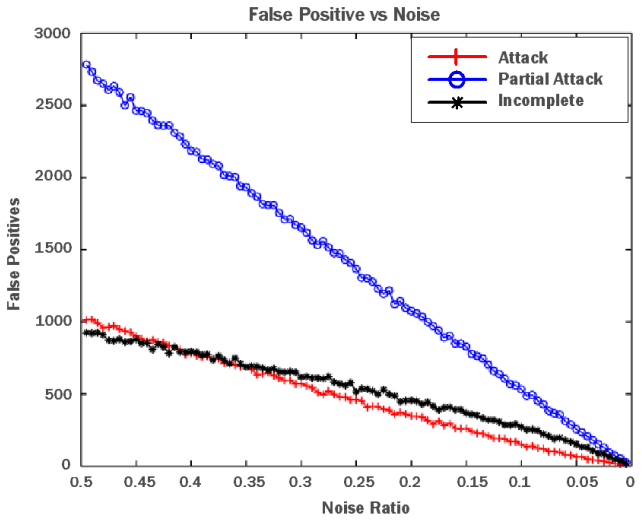

Fig. 11. Number of $False^+$ for changing noise ratios over 10,000 actions. A $False^+$ means an alarm is generated while there is no attack, partial attack or incomplete attack tree. $False^+$ are due to noise in input messages. Noise ratio is the ratio of attack actions to random actions.

Although preferred, $False^+$ limit the efficiency of the system. $False^-$ in turn are generated when attack trees are incomplete. $False^-$ require offline PC (Principle Component) or SVD (Singular Value Decomposition) like analysis on history of messages along with human interaction and feedback since there may not be any evidence of attack in the system.

Fig. 11 presents the result of the simulation. For each noise ratio (ratio of attack actions to random actions - probability that a randomly selected action belongs to an attack), we have calculated number of $False^+$ for a sequence of 10,000 actions. We have made multiple runs and take average of the results. Processing module successfully detects the correct attacks, subattacks and incomplete attack tree cases for the attack sequences randomly planted into input stream. Fig. 11 shows $False^+$ results of attack, partial attack and incomplete tree alarms for noise ratios $0.5, 0.45, 0.40, \ldots, 0.0$. As the noise ratio increases in the message log, number of $False^+$ increases as expected.

VIII. Conclusion

In this paper, we introduce a powerful technique to represent and detect complex attacks. The contribution of this work is threefold. First, we introduce the notion of *complex attacks*. Second, we model complex attacks using formal methods. Third, we show how to detect them. Our technique advances the IDS approaches to capture complex attacks such as insider and stealth attacks. We discuss how to realize the proposed technique in the context of IEEE 802.11 WLAN security. Simulation results show that it successfully detects attack sequences in large noisy input message streams.

References

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

[2] M. Jakobsson, S. Wetzel, and B. Yener, "Stealth attacks on ad-hoc wireless networks," in *IEEE Vehicular Technology Conf.*, 2003, pp. 2103–2111.

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

[4] 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/

[5] J. Steffan and M. Schumacher, "Collaborative attack modeling," in *ACM Symposium on Applied Computing*, 2002.

[6] S. Convery, D. Cook, and M. Franz, "Bgp attack tree," The Internet Engineering Task Force, 2001.

[7] K. Daley, R. Larson, and J. Dawkins, "A structural framework for modeling multi-stage network attacks," in *International Conference on Parallel Processing Workshops*, 2002.

[8] J. Dawkins and J. Hale, "A systematic approach to multi-stage network attack analysis," in *IEEE International Information Assurance Workshop*, 2004.

[9] J. McDermott, "Attack net penetration testing," in *New security paradigms workshop*, 2000.

[10] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *New Security Paradigms Workshop*, 1998.

[11] S. Noel, E. Robertson, and S. Jajodia, "Correlating intrusion events and building attack scenarios through attack graph distances," in *20th Annual Computer Security Applications Conference*, 2004.

[12] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Computer Security Foundations Workshop*, 2002.

[13] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *IEEE Symposium on Secuirty and Privacy*, 2002.

[14] G. Vigna, S. Eckmann, and R. Kemmerer, "Attack languages," in *IEEE Information Survivability Workshop*, 2000.

[15] G. Vigna, S. T. Eckmann, and R. A. Kemmerer, "The stat tool suite," in *DARPA Information Survivability Conference and Exposition*, 2000.

[16] S. Kumar, "Classification and detection of computer intrusions," Ph.D. dissertation, Purdue University, August 1995.

[17] J. May, J. Peterson, and J. Bauman, "Attack detection in large networks," in *DARPA Information Survivability Conference and Exposition*, 2001.

[18] W. Li, L. Zhi-tang, and W. Qi-hong, "A novel technique of recognizing multi-stage attack behaviour," in *Networking, Architecture, and Storages*, 2006.

[19] S. Cheung, U. Lindqvist, and M. W. Fong, "Modeling multi-step cyber attacks for scenario recognition," in *DARPA Information Survivability Conference and Exposition*, 2003.

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

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

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

[23] 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.

[24] G. Golub and C. Loan, *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press, 1996.

[25] S. A. Camtepe, M. Krishnamoorthy, and B. Yener, "A tool for internet chatroom surveillance," in *Second Symposium on Intelligence and Security Informatics*, 2004.

[26] E. Acar, S. A. Camtepe, M. Krishnamoorthy, and B. Yener, "Modeling and multiway analysis of chatroom tensors," in *IEEE International Conference on Intelligence and Security Informatics*, 2005.

[27] W. G. IEEE, "802.11," IEEE 802.11, 2005. [Online]. Available: http://grouper.ieee.org/groups/802/11/

[28] 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.

[29] 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.

[30] A. T. Rager, "Wep crack and airsnort," SourceForge, 2005. [Online]. Available: http://wepcrack.sourceforge.net/