Constrained Automata: a Formal Tool for ICT Risk Assessment

F.Baiardi [a], F.Martinelli[b], L.Ricci[a], C.Telmon[a]
[a]*Dipartimento di Informatica, Università di Pisa, Italy*
[b]*Istituto di Informatica e Telecomunicazioni, CNR, Pisa, Italy*

Abstract Conditional security assesses the security of an ICT system in a specifc context. A fundamental step of this assessment determines the threats that can implement an attack against the system. Constrained attack automata are finite state automata to formally conducting this step by decomposing complex attacks into a sequences of elementary attacks. Each state of the automata corresponds to a set of resources controlled by the attacker while a while final states correspond to the success of a sequence of attacks so that one threat has reached one of its goals. Each transition is paired with some constrains on the amount of computational resources, the skills and the knowledge required to implement the elementary attack. To exploit these automata, each threat is modeled in terms of the amount of computational resources, skills and knowledge that it has available and this amount is modelled as a tuple of elements of partially ordered sets. By comparing the amount of resources a threat can access against that required by an attack, we can determine if there is at least one threat that can implement the attack and available countermeasures. We also consider risk mitigation the application of a set of static countermeasures or of dynamic ones. A static countermeasure prevents a threat from exploiting a vulnerability and it is modeled by removing some automata transitions. Lastly, we discuss redundant countermeasures and how constrained attack automata can model dynamic countermeasures, i.e. actions that are executed as the attack is going on to stop the attack itself.

Keywords. Attack, state automata, threats, countermeasure, redundancy

Introduction

While a large amount of attention has been paid to formal models for unconditionally security, i.e. the ability of a system or of a component to withstand any attack, less attention has been paid to conditional security, i.e. to evaluate whether a system can withstand the attacks that can occur in a given context only [4-6, 14, 25, 26, 29, 33]. The goal of conditional security is a better return on the investment, **roi**, because it is focused only on the attacks that may occur in the considered context. From an operational point of view, the evaluation of conditional security corresponds to the risk assessment of the target system TS. This assessment should determine:

1. the vulnerabilities of the target system [1, 2, 15, 17, 22],
2. the attacks enabled by these vulnerabilities,
3. the threats that can implement these attacks in the considered context,
4. the attacks that may occur in the context,

5. the impact of attacks, i.e. the losses due to successful attacks,
6. the countermeasures that may be adopted either to prevent the success of an attack or to reduce the impact of successful attacks.

A threat is a source of attacks: physical events such a storm or a flooding, legal or illegal users of the system are a few examples of possible threats that results in distinct attacks to the system. However, in the following we neglect all attacks due to physical events and focus on those implemented by human beings. An important step of the assessment determines the possible threats, together with the resources available to each threat, in the considered context. We use resources in a fairly broad sense ranging from computational resources to skills or information on the system architecture. Step d) of the assessment merges the information about the threats and the attacks to define the attacks that may occur in the context. The last step of the assessment, risk mitigation, chooses a set of countermeasures, i.e. mechanisms and policies, to either prevent some attacks or to minimize their impacts on the target system. To achieve a satisfactory roi, the countermeasures are defined with reference to the attacks returned by step d) only, the only ones that may be successful in the considered context.

The matching of threats with the attacks they can implement against TS is one of the focuses of this paper, that introduces constrained attack automata as formal tool to support the matching. A constrained automaton CA(TS, T) is a finite state automaton that models the attacks that can be implemented by the threat T against TS as a sequence of state transitions, each corresponding to a elementary attack. With respect to traditional automata, constrained automata take into account the resources an attack requires, so that a state transition occurs if and only if T can access the resource to implement the corresponding elementary attack. To define CA(TS, T), T is modelled in terms of the goals it is trying to achieve, of the resources it can control as well as of risk aversion, i.e. the attitude of T with respect to possible prosecution. To this purpose, we introduce a distinct poset P(Kr) for each kind Kr of resources that attacks require. The elements of P(Kr) represents distinct levels of availability of Kr. Hence, n distinct kinds of resources are modelled by n posets $P(Kr_1)$, …, $P(Kr_n)$ and a threat T will be modelled as a tuple $<ra_1, \ldots, ra_n>$, where each ra_i belongs to P(Kri). An elementary attack A is modelled in terms of posets too because it is represented by a tuple $<rr_1, \ldots, rr_n>$ where each rr_i belongs to $P(Kr_i)$ and it defines the amount of the resource that is required to implement A. A threat T can implement the attack A only if each value of $<ra_1, \ldots, ra_n>$is not smaller than the corresponding one of $<rr_1, \ldots, rr_n>$. n is fixed for the assessment and it depends upon the detail level of the assessment.

We model the goals of a threat T as a subset SR(T) of the resources of TS. In this way, we can deduce the attacks T is interested in because each successful attack enables T to control a subset of the resources of TS. In turn, this implies that each state S of the automata may be mapped into the resources R(S) that a threat control after executing the attacks leading to S. Hence, T is interested in executing the attacks leading to state S only if interested in controlling R(Fs), that is if $R(Fs) \subseteq SR(T)$.

Sect. 1 discusses the modeling of threats and of elementary attacks in terms of the resources they, respectively, control and require. Attack automata and constrained attack automata are introduced in Sect. 2. The risk mitigation step and the definition of attack countermeasure is discussed in Sect. 3. Each countermeasure may consist in a new component or in a control that prevents the success of an elementary attack. This section formally defines the notion of a complete set of countermeasures i.e. a set of countermeasures that can stop any attack against the target system. Formally, the definition of a complete set of countermeasures for a constrained attack automaton prevent some transitions in the automata and it may be described as a cut set of a graph defined in terms of the automaton. Lastly, we define a k-redundant set of countermeasures, where k>1, defined a set of countermeasures that can prevent an attack even if at most k of its controls are faulty.

The notion of constrained attack automaton is inspired to that of attack graph [1, 9, 12, 16, 18, 23, 24, 27, 30, 34, 35] and several concepts are similar in the two frameworks. The main difference is that automata do explicitly model the order in which elementary attacks are executed while a graph may state that some attacks are required before a further one can be implemented but does not need to specify the execution order of such attacks. From this point of view, attack graphs are similar to And/Or attack tree [9, 17, 28] because they define the decomposition of complex attacks into elementary ones without constraining the execution order of elementary attacks. From our point of view, the order of attacks is important when defining the countermeasure of attacks. As attack graphs, attack automata may be exploited both in the planning of attacks or of countermeasures as well as in the analysis of information returned by a set of sensors to discover attacks that are currently going on against the system [7, 10, 11, 14, 20, 21, 26]. However, attack graphs have never been considered in the framework of modelling resources available to the various threats [33].

1. Modelling Threats and Elementary Attacks through Posets

This section discusses the modelling of elementary attacks and threats as tuples of elements of a poset. The model does not fix the number of elements in a tuple because it depends upon both the considered assessment and the detail level of the assessment. We discuss at first the modelling of threats in terms of poset and then the modelling of attacks. In the following, we neglect threats such as a flooding or an earthquake.

1.1. Threat Modelling

A threat can successfully implement an attack provided that it can access the resources the attack requires and it is willing to accept the risk of being discovered. Each feature may be modelled in terms of a partially ordered set, a set of elements and a partial order among the elements. In the following, the poset $<SA, <_{sa}>$ and the poset $<SR, <_{sr}>$ describe, respectively, the resources available and the risk aversion of the threats. In security, posets have been often be applied to model access rights [4].

At first, we consider SA and $<_{sa}$. Any element of SA corresponds to a distinct amount of resources. If a_1 and a_2 belong to SA and $a_1 <_{sa} a_2$, then the amount of resources modelled by a_1 is smaller than the one modelled by a_2. Smaller means that it can be collected more easily or, from another point of view, that is available to a larger number of people. The adoption of partial order make it possible to consider also cases where some of the elements cannot be compared. As an example, this happens if the resources are skills and abilities. Two attacks could require, respectively, skills and abilities in distinct languages and ordering these skills and abilities is not only rather complex but also inappropriate.

Each element of SR corresponds to a risk the threat is willing to accept. An element of SR is larger than another if it corresponds to the acceptance of a larger risk. In other word, if two threats are paired with, respectively, r_1 and r_2 and $r_1 <_{sr} r_2$, then the threat paired with r_2 is willing to accept a risk larger than the other one when executing an attack.

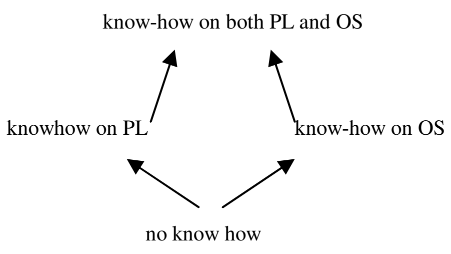
In general, one poset does not support an accurate evaluation of the resources available to a threat. The accuracy increases if distinct kinds of resources are modelled through distinct posets. Hence, in the most general case, each threat *T* will be characterized by a tuple *Tu(T)* of n, n>1, values. The first n-1 values of *Tu(T)* describe the resources *T* can exploit in its attack, while the last one describes the attitude of *T* towards risk. Larger values of this element model larger risks T is willing to accept.

Suppose, as an example, attacks against an application written in a language PL and running on an operative system OS. In a first, simple, case, we can model the knowledge required by an attack through the four elements poset shown in Fig. 1a. If a more detailed modelling is required, then we can adopt the poset Fig. 1.b that distinguish among distinct levels of knowledge of either or both topics. An even more complex solution adopts two posets to model the two know hows.

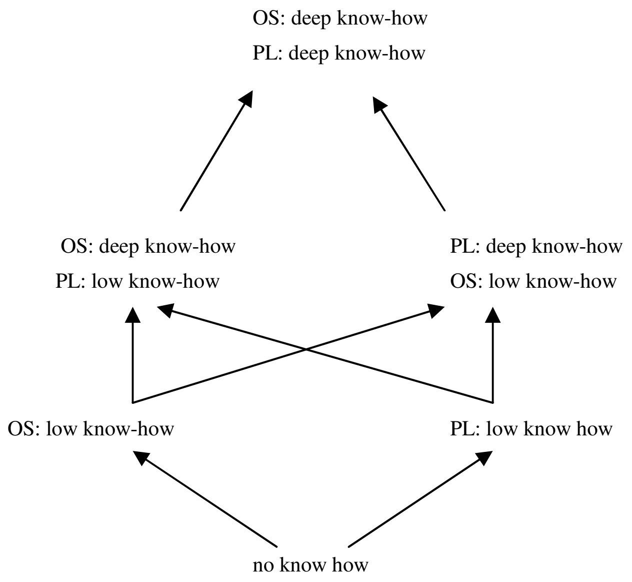
A further issue to be considering when modelling a threat T is the goals of T. In the following we assume that each goal corresponds to a distinct subset of the resources of the target system. In the most general case, the resource available to a threat and the risk aversion of the threat are a function of its goals. If T has several alternative goals, then resource availability and risk aversion change according to the goals. Therefore, in the following we decompose T into a set of "virtual" threats, each with just one goal, that is one of the goals of T. To interpreter the results of the assessment for T, we merge all the results for the virtual threats resulting from the decomposition of T. In the following, threat denotes a virtual threat.

2.2 Attack Modelling

The first step of attack modelling decomposes each attack into one or more sequences of elementary attacks. An *elementary attack* does not need to be further decomposed because it involves the application of *just one exploit*. The decomposition will be described in more details in the next section. Here we are interested in the description of an elementary attack in terms of both the resources it requires implement and the corresponding risk. An elementary attack is modelled in the same way of a threat, that is through a tuple of values, one for each poset. All the considerations for threats applies to attacks as well. Notice that each poset should include all the elements required to model both any attack and any threat. Hence, the definition of a poset should consider both all the threats and all the elementary attacks. In turns, this implies that an iterative refinement of the posets may be required as new attacks and/or new threats are taken into account.



a) A poset modelling the know-how of a threat



b) A more detailed modelling the know-how of a threat

Figure 1 Alternative posets to model a threat

Let us assume that the elementary attack *ea* and the threat t are paired, respectively, with the tuples *resreq(ea)* and *resav(t). The attack ea is **feasible** for T* if and only *if each element of resreq(a) no larger than to the corresponding element of resav(T).* The notion of feasible attack for a threat resumes the constrains on the resources required to successfully implement an attack. In the following we assume that the resources used for an elementary attacks may be reused for a further elementary attack.

2. Attack Automata and Constrained Attack Automata

This section introduces the notion of attack automata that describes one or more complex attacks, i.e. one or more sequences of elementary attacks, against the target system. Then, the notion of attack automaton evolves into that of constrained attack automaton. While an attack automaton is always associated with a target system, two kinds of constrained automata will be introduced that are associated with, respectively, a single threat and the target system.

2.1. Attack Automata

An attack automaton AA(TS) is a, nondeterministic or deterministic, finite state machine that models attacks of any complexity against the system TS. To this purpose:

a) each initial state of the machine corresponds to a state of TS where no attack has been executed yet,
b) each state *fs* of the machine corresponds a set of resources controlled by an attacker, let such resources be denoted by *rcs(fs)*,
c) each elementary attack *at* is paired with a unique label *la(at)* and each component *c* with a unique label *la(c)* so that each transaction is paired with a pair of labels to describe the attack and the component that is attack target ,
d) if *la* is the label of the attack *att(la)* then *resreq(la)* is the n-tuple that describes the resources to implement *att(la)*,
e) if there is a transition from a state s_1 to a state s_2 then $rcs(s_1) \subseteq rcs(s_2)$ because each attack cannot decrease the number of resources controlled by the attacker. This corresponds to the monotic property of attack graphs [34].

The mapping of each state into the resources the threat controls is a generalization of [8] that pairs states and access privileges. A complex attack that results in the control of a set SR of resources by the attacker is described as a sequence of transitions from an initial state to the one corresponding to the control of resources in SR. By pairing each transition with both an attack and a component, automata can easily model instances of the same attack against distinct components because they will be associated with the same attack label. In general, two attacks are instances of the same one if they exploit the same vulnerability and the same attack mechanisms in distinct instances of the same components. The existence of a label for the component allows us to model cases where vulnerabilities has been removed from some but not all the instances of the components used in the target system. This simplifies the analysis with respect the case where a transaction is labelled by a vulnerability only.

Formally, an attack automaton is a tuple <S, Is, Fs, Al, Co, Tr, Rcs > where

i) S is the set of the states,
ii) Is is the set of inital states , $Is \subseteq S$,
iii) Fs is the set of final states, $Fs \subseteq S$, $Is \cap Fs = \emptyset$,
iv) Al is the set of attack labels,
v) Co is the set of component labels,
vi) Tr is the set of transitions, $Tr \subseteq S \times S \times Al \times Co$,
vii) Rcs maps a state into the resources controlled by the attacker. It satisfies a non decreasing condition so that for $<s1, s2, a, c> \in Tr$ then $rcs(s1) \subseteq rcs(s2)$.

Tr is different from $S \times S \times Al \times Co$ because an attack may be ineffective in some state. Initial and final states of the automata are disjoint because in an initial state a threat cannot control the resources it is interested in. Alternative distinct initial states support the modelling of attacks of distinct threats. Consider, as an example, an attack of a user of the target system, i.e. to increase the available privileges, vs the attack of someone that does not have an account on the target system. The two attacks begin in distinct states of the target system. In the following, we denote the I-th element of AA(TS) by AA(TS)/i , i∈ 1..7.

An important property of attack automata is that they are acyclic, i.e. a sequence of transition cannot visit the same state twice or a sequence of attacks cannot lead to one of the states that has already been visited. If we consider that each automaton state corresponds to a set of resources controlled by the attacker, any cycle includes at least one useless attack that cannot increase the resources the threat already controls.

As already recalled, AA(TS) describes all the attacks against TS, independently of the existence of a threat that can implement them. The next section describes how to simplify the automaton when considering the resources available to threats.

2.2. Constrained Attack Automata

This section introduces a two steps procedure to build the automaton that describes all the attacks of a given set of threats against the target system. The first step builds, for each threat, an automaton describing all the attacks the threat can implement. Then, the second step merges all these automata into a single one.

2.2.1. Automaton associated with a Threat

CAA(TS, T), the *constrained attack automaton* associated with TS and with a threat T, is an attack automaton that describes the attacks T can implement against TS only. CAA(TS,T) is a subset of AA(TS) because it has the same number of elements of AA(TS,T) and each element of CAA(TS,T) is included in the corresponding one of AA(TS, T). The transformation of AA(TS) into CAA(TS, T) removes

1. the states T cannot reach or is not interested in reaching
2. the transitions that cannot occur because T cannot implement the corresponding attacks due to resource constrains.

Let us consider, at first, the states removed from AA(TS). They are removed in a forward step and in a backward one. The forward step removes from AA(TS) all the initial states that T cannot use to begin an attack and all those that can be reached from a previously removed state only. Then, the backward step removes all the final states that do not correspond to any goal of T and all the states that leads to these states only.

To formally define this, we introduce two sets, R_i and R_f, that include, respectively, the initial states of AA(TS) that T cannot use and the final states that T is not interested in reaching. Then, we remove

1. the states in $R_i \cup R_f$ from the states in AA(TS)/1,
2. the states in R_i from those in AA(TS)/2,
3. the states in R_f from those in AA(TS)/3.

We recursively remove now any state that either cannot longer be reached from the states in AA(TS)/2-Ri or that does no lead to a final state.

We now remove from AA(TS) the transitions corresponding to attacks requiring resources not available to T. T can implement any attack corresponding to the transition <s_1, s_2, *la*, *c*> only if any element of *resreq(la)* is not larger than the corresponding element of *resav(T)*. Any transaction violating this condition corresponds to an attack not feasible for T that should removed from AA(TS) together with all the states that cannot be reached from the initial states and those that do no leads to a final state. Any remaining state can be reached because T can

- starts any of its attacks in any initial state
- implement any attack corresponding to a transaction of CAA(TS,T).

Furthermore, each state that is not final leads to at least one final state.

In a more formal setting, the constrains on the resources available to, and the risk aversion of, T define a subset of attack labels to be removed from AA(TS)/4 and of transactions with such labels to be removed from AA(TS)/7. Then, any state that either is unreachable or does not lead to a final state is removed from AA(TS)/1 and AA(TS)/3.

If CAA(TS,T) does not include at least one final state of AA(TS), i.e. all the final states of AA(TS) have been removed or CAA(TS,T)/3 is empty, then any attack of T will be unsuccessful because it does not allow T to reach the final states it is interested in. Notice that this does not imply that T will not attempt to attack TS but rather that these attacks do not allow T to reach its goals. Hence, the assessment may neglect T. Obviously, the assessment terminates if no threat can implement a successful attack against TS.

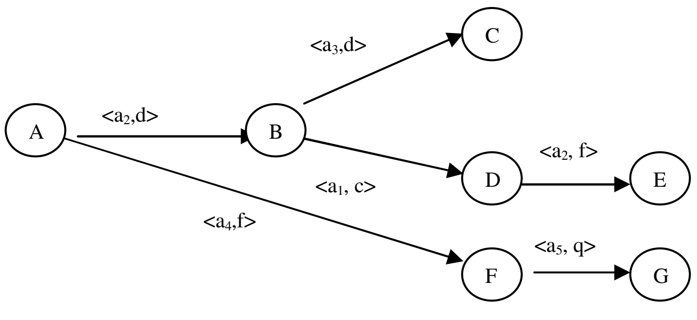

$$\begin{tikzcd}
 & & C \\
A \arrow[r, "{<a_2,d>}"] \arrow[rrd, "{<a_4,f>}"'] & B \arrow[ru, "{<a_3,d>}"] \arrow[r, "{<a_1, c>}"'] & D \arrow[r, "{<a_2, f>}"] & E \\
 & & F \arrow[r, "{<a_5, q>}"] & G
\end{tikzcd}$$

Figure 2. A subset of an attack automaton

Consider, as an example, the poset in Fig. 1b) and a threat T that can access a know how corresponding to element (OS deep, PL low) of the poset. Furthermore, let AA(TS) include the states represented in Fig. 2 that also shows all and only the transactions among the considered states. C, E and G are the final states that enable, respectively, the control of R1, R2 and R3. If T is not interested in controlling R2, then both E and D do not belong to CAA(TS,T) because T is not interested in the resources in Rcs(E) while D leads to E only. Furthermore, these states do not belongs to CAA(TS,T) also if T cannot access the resources to implement the attack a_1 paired with the transaction from B to D. As an example, this happens if the attack requires an amount of resource corresponding to the maximum of the poset in Fig. 1.b. If T cannot implement a_2, paired with the transaction from A to B, then all the states in Fig. 2 are eliminated when transforming AA(TS) into CAA(TS, T).

2.2.2. Automaton Associated with the Target System

CAA(TS), the constrained attack automaton associated with TS, is produced by merging the constrained automata CAA(TS, T) for any threat T. This implies that a state *st* of AA(TS) belongs to CAA(TS) iff there is at least one threat T such that *st* belongs to CAA(TS,T). In the same way, a transaction *tr* of AA(TS) belongs to CAA(TS) only if it belongs to CAA(TS,T) for some T.

Consider again the automaton described in Fig. 2 and assume that T_1 can implement the attacks a_2 and a_3 from A to C, while T_2 can implement these attacks as well as a_1 and a_2 from A to E. Lastly, no threat can implement the attack a_5 from F to G. Fig. 3 shows CAA(TS, T_1), CAA(TS, T_2) and CAA(TS).

CAA(TS) describes the attacks that can be successfully implemented taking into account the threats, the elementary attacks, the risk aversion of each threat, the resources each attack requires and those available to a threat. This is the minimal attack automaton to be considered by the assessment because no transition or no state may removed from the automaton without losing information on attacks that may occur.

3. Risk Mitigation

This step introduces a set of countermeasure for the attacks modeled by CAA(TS). We do not detail what a countermeasure is, in general, it is any security mechanism or policy that can prevent the successful implementation of an attack. It may consist in

- a new component that replaces one where a vulnerability has been detected,
- a set of checks to discover an attack and prevent it successful execution,
- a new component that prevents the threat from exploiting the vulnerability.

Countermeasures introduce a further constraint on the attacks paired by the same label because they are modelled as instances of the same attack, namely the existence of at least one countermeasure that prevents all the attacks and that can applied to any component affected by the vulnerability. If such a countermeasure does not exist, then the attacks are not instances of the same one and the analysis should be repeated after

- splitting the attacks paired with the same label into disjoint subsets and
- pairing each subset with a distinct label.

As any other component, a countermeasure may be unsuccessful because of static or dynamic faults. To take suh a failure into account, redundancy can be introduced to further increase system robustness. In the following, we consider independent countermeasures only, i.e. we assume that a failure in one countermeasure does not influence any other countermeasure.

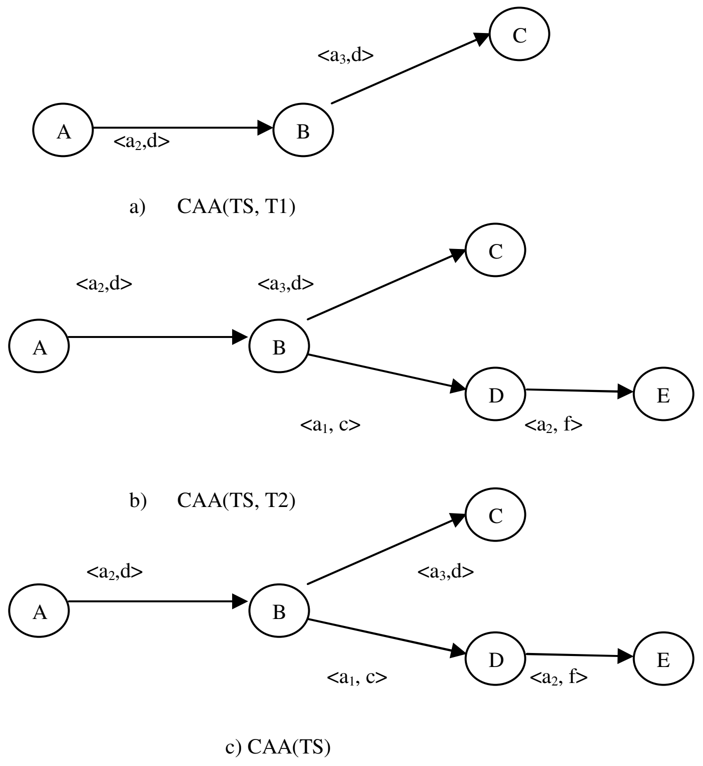


Figure 3. Constrained attack automata for distinct threats and the resulting automaton

3.1. Constrained Attack Automata and Countermeasures

To model risk mitigation, that is the application of countermeasures, or defensive actions [12, 23, 24], we remove from a constrained attack automata the transitions paired with attacks prevented by the countermeasures. This section is focused on static countermeasures, defined as those that remove a vulnerability before attacks occur and that stop any attack occurring after the risk mitigation step. As defined in the next section, a dynamic countermeasures can be applied as the attack is going on only.

Consider an attack a_1 associated with a transition of CAA(TS), if can apply a countermeasure for a_1., no threat can successfully execute a_1 against c, hence $< s_1, s_2, a_1, c>$ for any s_1 and s_2, cannot belong to CAA(TS). Let *Acm(CAA(TS))* be the set of pairs $< a_1,c>$ such that the countermeasure for a_1 has been applied to c. We are interested in a complete set of countermeasures *Cocm(CAA(TS))*, or in the critical set of attacks [12, 23, 24, 34], that is in a set of pairs such that after removing the corresponding transactions, CAA(TS) cannot reach any final state. A set of countermeasure is minimal if it is complete and none of its subset is complete.
In the following we denote by Cocm(TS) a complete set of countermeasure and neglect it depends upon CAA(TS). To characterize Cocm(TS), we consider AG(TS), the labelled directed graph that describes the automaton of CAA(TS). AG(TS) includes a distinct node *n(s)* for each state *s* of AG(TS). If *s* is an initial (final) state, then *n(s)* is an initial (final) node of the graph. Furthermore, AG(TS) includes a arc from $n(s_1)$ to

n(s₂) labeled by <*a, c*> if < s_1, s_2, *a, c*> belongs to CAA(TS). AG(TS) is acyclic because CAA(TS) is acyclic and it includes at least one path from an initial node to a final one because if no such path exists, then no countermeasure is required.

We recall that a set of arc CS(G) of a graph G is a cut set of G if, by removing all the arcs in S, no final node can be reached. A cut set is minimal if none of its subsets is a cut set. Since any set of countermeasures Cocm(TS) removes from AG(TS) all the arcs in A(Cocm(TS)) labeled by elements in Cocm(TS). We have that

- Cocm(TS) is complete iff A(Cocm(TS)) is a cut set of AG(TS)
- Cocm(TS) is minimal iff A(Cocm(TS)) is a minimal cut set of AG(TS).

In the graph in Fig.4, where A and H are the initial states and C, E and G the final ones, the set of countermeasures for <a_1, c_1> and <a_1, c_2> is a complete one because by removing the corresponding arcs, no final state can be reached. It is not minimal because the property holds even if we do not remove the arc <a_1,c_2>. Another complete, but not minimal, set includes the countermeasures for <a_2,c_3>,<a_3,c_4>, <a_1,c_2>. The set of countermeasures for <a_1,c_2>, <a_3,c_4>, <a_3,c_3> defines a minimal and complete set for the graph in Fig.5, because none of its subset is a cut set. A further complete and minimal set includes the countermeasures for <a_3,c_3> and <a_1,c_1>.

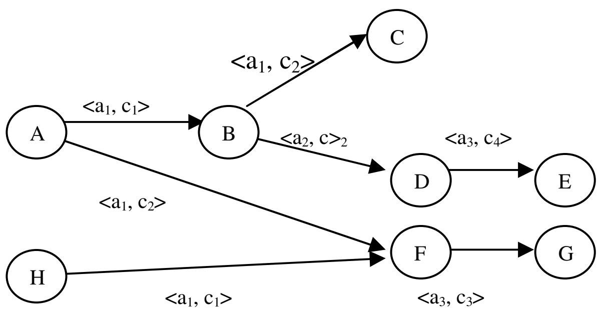


Figure 4. A CAA(TS)

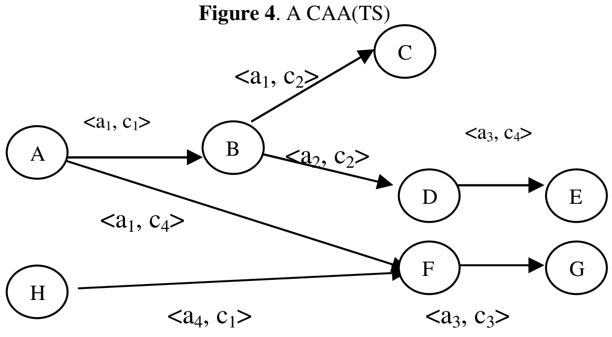


Figure 5. Complete and minimal sets of countermeasures.

A complete set of countermeasure prevents the successful execution of any attack because no final state can be reached after removing the elementary attacks prevented by the countermeasures. A set of countermeasures is minimal if one final state can be reached if any of its countermeasures is not applied. Notice that a minimal set of countermeasure does not define, in general, a minimal cut set of the attack graph because any time we introduce a countermeasure for an attack labeled by *la* and apply it to a component *c*, this removes all the arcs labeled <*la,c*>. Only if the set of countermeasure is optimal, the cut set is a minimal one.

In terms of the automaton, we have that a set of countermeasures is complete if for final state and any path, the set include at least one transition on the path.

Another important notion is that of redundant set of countermeasures. Such set may include several countermeasures to take into account that some of them could fail because of errors or fault in the implementation of the countermeasure. A set of countermeasure is k-redundant if can prevent any successful attack even if at most k of its countermeasures fail. As an example, a set of countermeasures is 2-redundant if it prevents any successful attack even if no more of two countermeasures fail. The set of countermeasure previously considered is a 0-redundant set. Since the failure of a countermeasure may be described as an arc that has not been removed from AG(TS), a k-redundant set of countermeasure can be defined as the union of k pairwise disjoint cut sets of AG(TS) so that if one arc is not removed because of the failure of a countermeasure, other countermeasures can stop the threat.

In a more formal setting, a k-redundant set of countermeasures is the union of CM_1, … CM_k, where for any $1 \leq i, j \leq k$

- Cmi is a complete set of countermeasures
- $CM_i \cap CM_j = \emptyset$.

To prove this consider that in the most general, and severe, graph all the arcs are labeled by a distinct pair <*a, c*> and no arcs on distinct paths from an initial state to a final one have the same label. A complete, and minimal, set of countermeasure CS can be defined by considering an arc <*a,c*> for each path and by including in CS a countermeasure for <*a,c*>. If an arc belongs to two sets of countermeasure, a final state can be reached if the corresponding countermeasure fails, hence the two sets do not define a k-redundant set for any $k \neq 0$. Hence, in general, an intersection between two sets of countermeasures reduces by one the degree of redundancy. As a consequence, a k-redundant set can be defined only if each path from an initial state to a final one includes at least k arcs with distinct labels. Shorter path prevents the definition of a k-redundant set because all the countermeasures for the attacks corresponding to the labels on the path may fail.

3.2. Dynamic Countermeasures

We consider now dynamic countermeasures, that is countermeasures that do not remove the vulnerability but try to prevent the evolution of the target system TS into a state where the threat achieve it goals. These countermeasures can be modeled as a set of actions to be executed to defend TS upon discovering that it has entered a given state. We assume the actions are executed by a defender that is by the system owner to prevent an attacker to control TS. As a consequence, the overall situation can be modeled by an automaton where some transitions occur because of an elementary attack, while other transitions are due to the defender actions. Obviously, the goal of threat is a sequence of transitions ending in a final state of the automaton, that of the defender is a sequence of transitions that returns TS to an initial state or at least that prevents TS from reaching a final state. Notice that some state can be paired with no action of the defender. This models the case where the defender has no visibility of the state, i.e. the defenders cannot know that TS has entered into the corresponding state. Notice that a state can be paired with a defender action provided that it is not a final one because final states model the success of the attack.

An *interactive automaton* describes the results of the actions of the attacker, i.e. of the threat, and of those of the defender. To define the automaton, we have to specify the sequence of elementary attacks to be executed starting from an initial state, the equivalence relation among states and the defender actions for the various classes. At each step, we consider the current state of the automata *cs* and the next elementary attack, *ea*, the first action of the attacker sequence of actions still to be considered. The actions of the attacker or of the defender are defined a priori, independently of the those of the opponent. The following rule is applied:

- if *cs* is not paired with an action of the defender, then *ea* is applied. This consumes the action, i.e. the action following *ea* in the sequence is considered
- if *cs* is paired with an action *ad* of the defender, then the automaton chooses in a nondeterministic way whether to execute *ad* or *ea*. If it chooses *ea,* then it enters a state where a distinct defender action will be considered. If, instead, it chooses *ad*, then *ea* is not consumed and it may be executed in the next state.

A further case is the one where the action of the attacker depends upon the considered state of the automaton. Now, the attacker actions are not known in advance because the i-th action depends upon the i-th state of the automaton. In this case:

- the attacker actions are a function of the state that has been reached by the automaton, an empty action is possible
- the defender actions may be specified for each state. An empty action is paired with a state that is no visible to the defender and with other states as well.
- in each state that specifies both an attacker action and a defender, a nondeterministic choice occurs
- for each initial state there is at least one sequence of attacker actions that leads the automaton in a final state
- in any initial or final state no action of the defender is possible.

Because of nondeterminism, the execution of the automaton may terminate in a set of states. The following cases may occur:

a) any state is a final one: this denotes a complete success of the attacker,
b) any state is an initial one: this denotes a complete success of the defender,
c) at least one state is final: this will be considered as a success of the attacker,
d) no set is final and at least one is initial: this will be considered as a partial success of the defender.

In case a), the actions of the defenders are ineffective because only final states are reached. The reverse is true in case b) because the target system is restored into a correct state. Case c) is the most interesting one where either a success or a failure of the attacks is possible according to the timing of the action. The last case is the most ambiguous one because the target system is left in a state that is not correct and where new attacks can be more effective.

Consider now an automaton where the execution ends in a set of states including at least one final state *fs*. We say that a state *s* is critical if an execution reaches *fs* because of a choice done in *s*. A state *s* belongs to *cs(fs)*, the critical set of a final state *fs*, if it is critical for at least one attack sequence. The critical set points out the states where the choice of the action to be executed influences the final results.

In order to automatize such analysis, we plan to model it as a module checking problem and apply the formal techniques for checking the behavior of systems in presence of several uncertain environments as specified in [12]. Ideally, we could model each environment (attacker) that induces an outcome of its interactions on the system (defender). With such techniques we can check all the possible outcomes (attacks vs countermeasures).

4. Conclusion

This work has presented some tools to support a formal approach to risk assessment. In particular, we have considered attack automata that support the modelling of complex attacks as alternative sequences of elementary attacks against a system component. To determine the attacks that can be actually be executed, posets are defined to evaluate the resources a threat can access and to compare these resources against those required to implement the attack. In this way, the automata that describe the attack against the considered target system can be simplified by removing those attacks that no threat can execute.

The adoption of static countermeasures can be formally described in terms of a cut set of a graph that describes the attack automaton. Dynamic countermeasures can be described as further state transitions besides those modeling elementary attacks.

The main problem still to be considered is the probability that an attack occurs and the corresponding risk. A correct evaluation of this probability requires the availability of information about the history of the system and not only formal tools for the assessment.

References

[1] P. Ammann , D. Wijesekera , S. Kaushik, Scalable, Graph-based Network Vulnerability Analysis, *9th ACM Conf. on Computer and Communications security*, Nov. 18-22, 2002, Washington, DC, USA

[2] W. A. Arbaugh, W. L. Fithen, J. McHugh , Windows of Vulnerabilits: a Case Study Analysis, *IEEE Computer*, December 2002, p.52 - 59.

[3] R. Baldwin, H.Kuang, *Rule Based Security Checking*, Technical Report, MIT Lab for Computer Science, May 1994.

[4] M.Bishop, *Computer Security*, Addison Wesley, 2003.

[5] CC-project, *Evaluation Methodology, Common Criteria for IT Security Evaluation*", CEM-99/045 Aug.1999.

[6] CC-project, *User Guide. Common Criteria for IT Security Evaluation*, Oct. 1999.

[7] F.Cuppens, A. Miège, Alert Correlation in a Cooperative Intrusion Detection Framework, *2002 IEEE Symposium on Security and Privacy*, p.202, May 12-15, 2002

[8] M.Dacier, *Towards Quantitative Evaluation of Computer Security*, Ph.D Thesis, Institute National Polytechnique de Tolouse, Dec 1994

[9] J. Dawkins, C. Campbell, J. Hale, Modeling Network Attacks: Extending the Attack Tree Paradigm, *Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, Johns Hopkins University, June 2002.

[10] C. W. Geib and R. P. Goldman, Plan Recognition in Intrusion Detection System, *DARPA Information Survivability Conference and Exposition (DISCEX II),* June 2001.

[11] R. P. Goldman, W. Heimerdinger, and S. A. Harp. Information Modeling for Intrusion Report Aggregation*, DARPA Information Survivability Conference and Exposition (DISCEXII)*, June 2001.

[12] Orna Kupferman and Moshe Y. Vardi, Module Checking, *8th Int. Conference on Computer Aided Verification, LNCS 110*2, p. 75-86, 1997.

[13] S. Jajodia, S. Noel, B. O'Berry, Topological Analysis of Network Attack Vulnerability, *Managing Cyber Threats: Issues, Approaches and Challenges*, Kluwer Academic Publisher, 2003.

[14] C. Lala, B. Panda, Evaluating damage from cyber attacks: a model and analysis, IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol 31 (4), July 2001, p.300- 310.

[15] U. Lindqvist, E. Jonsson, How to Systematically Classify Computer Security Intrusions, *1997 IEEE Symposium on Security and Privacy*, May 1997.

[16] K. Lye, J. Wing, Game strategies in network security, *Foundations of Computer Security Workshop*, July 2002.

[17] R. A. Martin, Managing vulnerabilities in networked system, *IEEE Computer*, November 2001. p. 32 - 38.

[18] F. Moberg, *Security Analysis of an Information System using an Attack Tree-based Methodology*, Master thesis, Chalmers University of Technology, 2000.

[19] P. Moore, R. J. Ellison, R. C. Linger, *Attack Modelling for Information security and Survivability*, Technical note CMU/SEI- 2001-TN001.

[20] P. Ning , P.Cui , D. S. Reeves, “Constructing attack scenarios through correlation of intrusion alerts”, Proc. of the 9th ACM conference on Computer and communications security, November 2002, Washington, DC, USA.

[21] P. Ning, D. Xu, C. Healey, R. .St. Amant, Building Attack Scenarios through Integration of Complementary Alert Correlation Methods, *11th Annual Network and Distributed System Security Symposium*, February, 2004.

[22] P. Ning, D. Xu, Hypothezing and Reasoning about Attacks Missed by IDSs, *ACM Trans. On Information System Security*, Vol.7, No.4, Nov. 2004, pp. 591-627

[23] R.W. Ritchey , P. Ammann, Using Model Checking to Analyze Network Vulnerabilities, *2000 IEEE Symposium on Security and Privacy*, p.156, May 14-17, 2000.

[24] S. Jha, O. Sheyner, J. M. Wing. *Minimization and Reliability Analyses of Attack Graphs.* Technical Report CMUCS-02-109, Carnegie Mellon University, February 2002.

[25] S. Jha , O. Sheyner , J. Wing, Two Formal Analysis of Attack Graphs, *15th IEEE Computer Security Foundations Workshop* ,p.49, June 24-26, 2002.

[26] C. Phillips, L. Painton Swiler, A graph-based system for network-vulnerability analysis, *Workshop on New Security Paradigms*, p.71-79, September 22-26, 1998.

[27] X. Qin, W. Lee, Attack Plan Recognition and Prediction Using Causal Networks, *20th Annual Computer Security Applications Conference* pp. 370-379, 2004

[28] R. Ritchey , B. O'Berry, S. Noel, *Representing TCP/IP Connectivity For Topological Analysis of Network Security*, 18th Annual Computer Security Applications Conference, p.25, Dec. 2002.

[29] B.Schneier, Attack Trees: Modeling Security Threats, *Dr. Dobb's Journal*, December 1999.

[30] O. Sheyner , J. Haines , S. Jha , R. Lippmann , J. M. Wing, Automated Generation and Analysis of Attack Graphs, *IEEE Symposium on Security and Privacy*, p.273, May 12-15, 2002 .

[31] O. M. Sheyner, *Scenario Graphs and Attack Graphs*, Ph.D. Thesis, CMU-CS-04-122, April 14, 2004.

[32] D. Smith, J. Frank, A.Jonsson, Bridging the Gap Between Planning and Scheduling, *Knowledge Engineering Review*, 15(1), 2000.

[33] L.P. Swiler, C. Phillips, D. Ellis, S. Chakerian, Computer-Attack Graph Generation *Tool DARPA Information Survivability Conference & Exposition* , June 2001.

[34] F.Swideriski, W.Snyder, *Threat Modelling*, Microsoft Press, 2003.

[35] S. J. Templeton , K. Levitt, A Requires/Provides Model for Computer Attacks, *Workshop on New security paradigms*, p.31-38, September 2000,

[36] S. Tidwell, R. Larson, K. Fitch, J. Hale, Modeling Internet Attacks, *IEEE Workshop of Information Assurance and Security*, June 2001.