

Constrained Automata: a Formal Tool for Risk Assessment and Mitigation

F.Baiardi¹, F.Martinelli², L.Ricci³, C.Telmon³

¹ Polo G.Marconi, Università di Pisa
Via dei Colli, La Spezia, Italy
f.baiardi@unispeszia.it

² Istituto di Informatica e Telecomunicazioni, CNR,
Pisa, Italy
fabio.martinell@itt.cnr.it

³ Dipartimento di Informatica, Università di Pisa
Largo B.Pontecorvo, 3, Pisa, Italy
(ricci, claudio) @di.unipi.it

Abstract: Conditional security assesses the security of an information and communication system in a specific context. A fundamental step of the assessment determines the threats of a system and the attack they can implement. Constrained attack automata are finite state automata to formally describe this step by decomposing complex attacks into sequences of elementary attacks. Each state of the automata corresponds to a set of components of the system controlled by the attacker while a final state models the success of a sequence of attacks that has enabled a threat to reach one of its goals. Each transition of the automata can occur provided that some constraints on the amount of computational resources, the skills and the knowledge required to implement the corresponding elementary attack are satisfied. To exploit these automata, each threat is modeled in terms of the amount of computational resources, skills and knowledge it can access. In turn, this amount is modeled as a tuple of elements where each element belongs to a partially ordered set. By comparing the amount of resources a threat can access against that an attack requires, we determine whether the threat can implement the attack. The attacks that can occur are an input of a risk mitigation step that defines static and dynamic countermeasures to be applied. A static countermeasure prevents the successful execution of an attack by removing a vulnerability and it is modeled by pruning some automata transitions. Instead, dynamic countermeasures are modeled as actions executed as the attack goes on to stop it. Lastly, we discuss redundancy to take into account error or fault in countermeasure implementation.

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

1. 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 that evaluates whether a system can withstand only the attacks that can occur in a given context [3-7, 15, 27, 28, 31, 35-38]. The goal of conditional security is a better return on the investment, **roi**, in security because it is focused only on the attacks that may occur in the current context of the target system. From an operational point of view, the evaluation of conditional security corresponds to the risk assessment of the

target system TS. This assessment includes several formal analyses that determine:

1. the vulnerabilities of TS [1, 2, 16, 19, 24],
2. the elementary attacks the vulnerabilities enable,
3. the threats interested in attacking TS in the considered context,
4. the complex attacks that the threats can implement,
5. the impacts of complex attacks, i.e. the losses due to successful attacks,
6. the countermeasures 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 flooding, legal or illegal users of the system are a few examples of possible threats that results in distinct attacks against the system. However, in the following, we neglect all attacks due to physical events and focus on goal oriented attacks implemented by human beings. An important step of the assessment determines the possible threats in the considered context together with the resources each threat can access. We use resources in a fairly broad sense ranging from computational resources to skills and/or information on the architecture and the implementation of TS. Step 4) of the assessment merges the information about the threats and the attacks to define the subset of attacks that may be successfully implemented in the context. The last step of the assessment, risk mitigation, chooses a set of countermeasures, i.e. security mechanisms and policies, to either prevent some attacks or to minimize their impacts on TS. To achieve a satisfactory roi, the countermeasures should be defined with reference to the attacks returned by step 4), the only ones that may be successful in the considered context.

The matching of threats with the attacks they can implement against the target system TS is one of the focuses of this paper that introduces constrained attack automata as formal tool to implement the matching. A constrained automaton CA(TS, T) is a finite state automaton that models attacks of any complexity a threat T can implement against TS as a sequence of state transitions, each corresponding to an

elementary attack. With respect to traditional automata, constrained ones take into account the resources an elementary attack requires, so that a state transition occurs if and only if T can access the resources to implement the corresponding elementary attack. To define $CA(TS, T)$, we model T 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. Distinct elements of the poset $P(Kr)$ represent distinct levels of availability of the resource modeled by Kr . Hence, the levels of availability of n distinct kinds of resources may be modeled through the Cartesian product of n posets $P(Kr_1), \dots, P(Kr_n)$ where n depends upon the considered assessment and the required accuracy. Assuming that the products has been defined, we can model through a tuple the resources a threat T can access and through another tuple those an elementary attack A requires, the i -th element of both tuples belongs to $P(Kr_i)$. T can implement A only if each value in the tuple that describes A is larger than or equal to the corresponding one in the tuple that describes T .

Since the paper is focused on intelligent threats that compose attacks against TS to achieve some goals, a goal of a threat T is modeled as the ability of controlling one or more security attributes of some resources of TS . In the following these resources will be denoted as component to distinguish them from those required to implement an elementary attack. In the simplest case, the set of these attributes should include, at least, confidentiality, integrity and availability but the set can be defined in a parametric way according to the assessment of interest. Starting from the goals of T , we can deduce the attacks T is interested in because each successful attack enables T to control a subset of attributes of the components of TS . In turn, this implies that each state S of automaton $CA(TS, T)$ may be mapped into $R(S)$ a set of component attributes that T controls. Hence, T is interested in executing the attacks leading to state S only if interested in controlling $R(S)$ that is if there is one goal of T equal to $R(S)$.

Sect. 1 of this paper 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. Sect. 3 discusses the risk mitigation step and the definition of attack countermeasures. Each countermeasure may consist in a new version of a component or in a control that prevents the successful execution of an elementary attack. This section formally defines a complete set of countermeasures i.e. a set of countermeasures that can stop any complex attack against TS . Formally, a complete set of countermeasures for a constrained attack automaton prevents some automata transitions and it may be described as a cut set of the graph that describe the automata transitions. Lastly, a k -redundant set of countermeasures, where $k > 1$, is a set of countermeasures that can prevent an attack even if at most k of its countermeasures fails because of errors or faults.

The notion of constrained attack automaton is inspired to that of attack graph [1, 10, 13, 17, 20, 25, 26, 29, 32, 36, 37] and several concepts are similar in the two frameworks. The main

difference is that automata do explicitly model the order of execution of elementary attacks while a graph may state that some attacks are required before a further one can be executed but it does not need to specify the execution order of these attacks. From this perspective, attack graphs are similar to And/Or attack trees [10, 18, 19, 30] because they define how a complex attack may be decomposed into elementary ones without constraining the execution order of these 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 [8, 11, 12, 13, 22, 23, 29]. However, attack graphs have never been considered in the framework of modeling resources available to the various threats [35]. [18] describes how distinct systems, or distinct versions of the same system, can be compared with respect to the set of attack strategies against a system. This approach is fairly complementary with the one we propose because it is focused on the system components and in the attacks they enable rather than on the resources an attack requires.

2. MODELING THREATS AND ATTACKS THROUGH POSETS AND TUPLES

This section discusses the modeling of elementary attacks and threats as tuples built out 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 modeling of threats in terms of poset and then the modeling of attacks. We recall that we neglect natural threats, such as flooding or an earthquake, and focus on intelligent threats with a predefined set of goals.

2.1 Threat Modeling

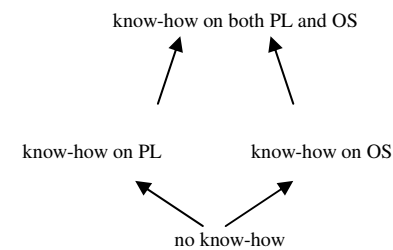
A threat can successfully implement an elementary attack provided that she can access the resources the attack requires and is willing to accept the corresponding probability of being discovered. Each feature may be modeled in terms of a partially ordered set, a set of elements and a partial order among the set elements. In the security field, posets are often applied to model access rights [5]. In the following, the poset $\langle SA, \langle_{sa} \rangle$ describes the resources available to the threats while $\langle RA, \langle_{sr} \rangle$ is the poset that describes the risk aversion of a threat.

At first, we consider $\langle SA, \langle_{sa} \rangle$ that describes the availability level of some logical or physical resource, i.e. each element of SA corresponds to a distinct amount of resources. If a_1 and a_2 belong to SA and $a_1 \langle_{sa} a_2$, then the amount of resources modeled by a_1 is smaller than the one modeled 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 threats. The adoption of a partial order makes it possible to consider also cases where some elements cannot be compared. As an example, this happens if the resources are skills and abilities.

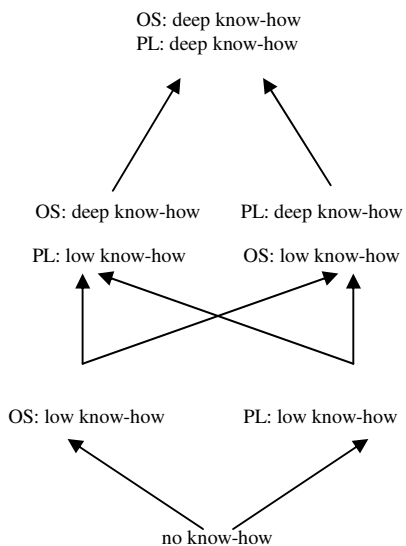
Two attacks could require, respectively, skills and abilities in distinct programming languages and ordering them is not only rather complex but also inappropriate.

In general, to model in an accurate way the resources available to a threat, several posets may be required. The accuracy increases if distinct posets model distinct kinds of resources. Hence, in the most general case, each threat T will be characterized by a tuple $Tu(T) = \langle ra_1, \dots, ra_n \rangle$ of $n, n > 1$, values. The first $n-1$ values of $Tu(T)$ describe the resources T can exploit to implement an elementary attack, while the last one describes attitude of T towards risk, the probability of being discovered T accepts. Larger values of this element model the acceptance of larger risks. Consider, as an example, attacks against an application written in a language PL and running on an operative system OS. A first, simple, assessment may model the knowledge to implement an elementary attack through the four elements poset shown in Fig. 1a. If a more detailed modeling is required, then the assessment can adopt the poset in Fig. 1.b that distinguish among distinct levels of knowledge of either or both topics. An even more accurate assessment may adopt two distinct posets to model the two know hows.

Any assessment uses the poset RA. Each element of this poset corresponds to a range of probabilities of being discovered the threat is willing to accept. An element r_i of



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



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

Figure 1 Alternative Posets to Model a Threat

RA is larger than an element r_2 if any value in r_1 is larger than one in r_2 . In other word, if two threats are paired with, respectively, r_1 and r_2 and $r_1 <_{ra} r_2$, then the threat paired with r_2 is willing to accept a probability larger than the other one of being discovered when executing an attack.

A further issue to be considering when modeling a threat T is the goals of T that is the goals of the complex attacks T implements. In the following, we assume that each goal corresponds in the control of a subset of the attributes of the resources of TS, TS components. As previously mentioned, these attributes include, at least:

1. *confidentiality*, the ability of accessing information in a component,
2. *integrity*, the ability of updating the information in a component,
3. *availability*, the ability of determining the other users that can access or update information in a component.

Other attributes, such as accountability, may be introduced according to the target system of interest and to the accuracy level of the assessment. Distinct goals correspond to distinct subsets of attributes. In the following, the goals of T are denoted as $G(T) = \{g_1(T), \dots, g_n(T)\}$ where each $g_i(T)$ is a subset of the security attributes of the components of TS. For each goal, an impact for the owner of TS may be defined as well. In the most general case, available resources and risk aversion of T are a function of the goal T is trying to achieve so that both resource availability and risk aversion depend upon the considered goal.

2.2 Attack Modeling

The first step of attack modeling decomposes each complex attack into one or more sequences of elementary attacks. The set of elementary attacks that may be implemented depends upon the vulnerability in the components of TS. An *elementary attack* does not need to be further decomposed because it exploits a single vulnerability or, from another point of view, it involves the application of *just one exploit*. However, it does not allow a threat to reach its goals. Hence, threats compose elementary attacks into complex ones according to the goal to be achieved. This planning of complex attacks will be described in more details in the next section because here we are interested in the description of an elementary attack in terms of both the resources its implementation requires and of the risk it poses to the threat executing it. An elementary attack is modeled in the same way of a threat that is through a tuple $\langle rr_1, \dots, rr_n \rangle$, where rr_i belongs to the corresponding poset. All the previous considerations for threats apply to attacks as well.

Since the same posets model both elementary attacks and threats, each poset should include all the elements to model with the required accuracy both any attack and any threat. In turns, this implies that the definition of each poset should consider any threat and any elementary attack. An iterative refinement of the posets may be required when the assessment consider further elementary attacks or threats.

Let us assume that each elementary attack ea and each threat T are paired, respectively, with the tuples $resreq(ea)$ and $resav(T)$. We say that *the elementary attack ea is feasible for T* if and only if *each element of $resav(T)$ larger than or equal to the corresponding element of $resreq(ea)$* . This notion formally states the constraints on the resources a threat T should be able to access in order to implement an elementary attack ea . In the following, we assume that the resources used for an elementary attack can be reused for further elementary attacks. Hence, a threat T can implement a sequence S of elementary attacks against TS provided that it can access those required by each attack in S .

3. CONSTRAINED ATTACK AUTOMATA

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

In the following we do not define in detail how attack automata are built. However, we notice that each threat can execute an elementary attack only if it controls some components attributes while the successful execution of an attack increases the set of attributes the threat controls. Hence, a complex attack can be described as a plan of a threat to increase the component attributes she controls till a goal is achieved, i.e. an attack plan composes elementary attacks to achieve sub goals that makes it possible to execute further attacks and so on till achieving the final goal. From another point of view, a threat can implement an elementary attack if it can access the proper resources and if it controls the proper set of attributes. An automaton describes how a threat can compose elementary attacks to control the attributes to execute further elementary attacks till reaching a goal, i.e. till controlling the proper set of component attributes. Hence, any planning algorithm can be adopted to discover complex attacks against a target system. However, an assessment is interested in determining *all possible plans* against the target system rather than a single one.

3.1 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:

- an initial state of the machine corresponds to a state of TS where no attack has been executed yet,
- each state fs of the machine can be mapped into a set $rscs(fs) = \{ \langle at, r_n \rangle \}$ where at is an attribute and r a component. The component attributes in $rscs(fs)$ are controlled because of the sequence of elementary attacks leading to fs ,
- an elementary attack at is paired with a unique label $la(at)$ and each component c with a unique label $la(c)$ so that each transition is paired with two labels that describe, respectively an elementary attack and the component that is the target of the elementary attack,

- if la is the label of the attack $att(la)$ then $resreq(la)$ is the n -tuple that describes the resources to implement $att(la)$,
- 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 component attributes the attacker controls. This corresponds to the monotone property of attack graphs [34].

The mapping of each state into the components the threat controls is a generalization of [8] that pairs states with access privileges. A complex attack that results in the control of a set SR of attributes by the attacker is described as a sequence of transitions from an initial state to the one corresponding to the control of components 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 all these instances will be paired with the same attack label. In general, two instances of the same attack exploit the same vulnerability and the same attack mechanisms against distinct instances of a component. The adoption of a label for the component allows us to model cases where vulnerabilities have been removed from some, but not all, the instances of a component of TS . This simplifies the analysis with respect to the case where a transition label considers the exploited vulnerability only.

Formally, an attack automaton is a tuple that belongs to the set $\langle S, Is, Fs, Al, Co, Tr, Rcs \rangle$ where

- S is the set of the states,
- Is is the set of initial states, $Is \subseteq S$,
- Fs is the set of final states, $Fs \subseteq S$, $Is \cap Fs = \emptyset$,
- Al is the set of attack labels,
- Co is the set of component labels,
- Tr is the set of transitions, $Tr \subseteq S \times Al \times Co$,
- Rcs maps a state into the component attributes the attacker controls. It satisfies a non decreasing condition with respect to transitions because if $\langle s1, s2, a, c \rangle \in Tr$ then $Rcs(s1) \subseteq Rcs(s2)$.

In the following, $AA(TS)/i$, $i \in 1..7$, denotes the i -th element of the tuple $AA(TS)$.

In general, the set of transitions Tr is a proper subset of $S \times S \times Al \times Co$ because an attack may be ineffective in some state. Initial and final states of an automaton are disjoint because in an initial state a threat cannot control the component attributes she is interested in. Alternative distinct initial states support the modeling of attacks of distinct threats. Consider, as an example, the complex attacks a user of TS can implement to increase his/her privileges vs the attacks of someone that cannot access an account TS . The two complex attacks begin in distinct states of TS that correspond to distinct sets of component attributes. The automaton includes a distinct final state for each distinct goal that at least one threat can achieve.

An important property of attack automata is that they are acyclic, i.e. a sequence of transitions cannot visit the same state twice or, in other words, a sequence of attacks cannot lead to a previously visited state. If we consider that each automaton state corresponds to a set of component attributes

the attacker controls, any cycle includes at least one useless attack that cannot increase the attributes the attacker controls.

As already recalled, AA(TS) describes all the attacks against TS, independently of the existence of a threat that can implement them. Sect.3.2 and 3.3 introduce a two steps procedure to build the automaton that describes all and only the attacks of a given set of threats against TS. 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. After defining static and dynamic countermeasures, Section 5 describes the modeling of risk mitigation through constrained automata.

3.2 Automaton Associated with a Threat

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

1. any state T cannot reach or is not interested in reaching because the attributes that T controls in these states do not match T goals;
2. any transition that cannot occur because T cannot access the resources to implement the corresponding attack.

Furthermore, an initial state s of AA(TS) is an initial state of CA(TS, T) as well if and only if T initially controls all the attributes in $Rcs(s)$.

Let us consider, at first, the states that are pruned from AA(TS). This pruning is implemented in two steps: a forward step and a backward one. At first the forward step considers any initial state s of AA(TS) that is not an initial state of CA(TS, T) and removes any state s that cannot be reached from an initial state of CA(TS, T). The forward step also removes all the states that can be reached only from a state that has been previously removed. Then, the backward step removes all the final states that do not correspond to any goal of T and all the states that lead to these states only.

To formally define the pruning, 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. any state in $R_i \cup R_f$ from AA(TS)/1,
2. any state in R_i from AA(TS)/2,
3. any state in R_f from AA(TS)/3.

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

We now prune from AA(TS) any transition corresponding to an attack that requires some resources that T cannot access. T can implement an attack that fires the transition $\langle s_1, s_2, la, c \rangle$ only if any element of $resreq(la)$ is not larger than the corresponding element of $resav(T)$. Any transition violating this condition corresponds to an attack a not

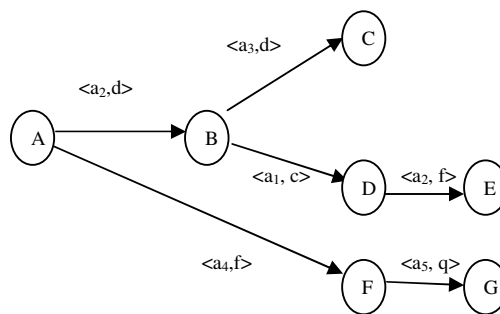


Figure 2. A subset of an Attack Automaton

feasible for T and that should be pruned from AA(TS) together with all the states that cannot be reached from the initial ones and those that do no leads to a final state.

To prove that any remaining automaton state can be reached consider that T can

- start a complex attacks in any initial state
- implement any attack corresponding to a transition of CA(TS,T).

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

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

If CA(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 CA(TS,T)/3 is empty, then any complex attack of T will be unsuccessful because it does not enable T to reach the final states T is interested in. Notice that this does not imply that T will not attempt to attack TS but rather that the elementary attacks of T cannot be composed into a plan that enables T to reach its goals. Hence, the assessment may neglect T. Obviously, the assessment terminates if no threat can implement a successful complex attack against TS.

Consider, as an example, the poset in Fig. 1b) and a threat T that can access a know-how corresponding to the element (OS deep, PL low). Furthermore, let AA(TS) include all the states and all and only the transactions in Fig. 2. C, E and G are the final states that enable, respectively, the control of the attributes of R1, R2 and R3. If T is not interested in controlling the attributes of R2, then both E and D do not belong to CA(TS,T). In fact, D leads to E only and T is not interested in the component attributes in $Rcs(E)$. Another case where these states do not belongs to CA(TS,T) is the one where T cannot access the resources to implement the attack a_1 paired with the transition from B to D. As an example, this happens if a_1 requires an amount of resource corresponding to the maximum of the poset in Fig. 1.b. If T cannot implement a_2 , the attack paired with the transition from A to B, then all the states in Fig. 2 are pruned when transforming AA(TS) into CA(TS, T).

3.3 Automaton Associated with the Target System

To build $CA(TS)$, the constrained attack automaton associated with TS , we merge the constrained automata in $\{CA(TS, T)\}$ where T is any threat that can implement a complex attack against TS . This implies that a state st of $AA(TS)$ belongs to $CA(TS)$ iff there is at least one threat T such that st belongs to $CA(TS, T)$. In the same way, a transition tr of $AA(TS)$ belongs to $CA(TS)$ if and only if it belongs to $CA(TS, T)$ for at least one threat T .

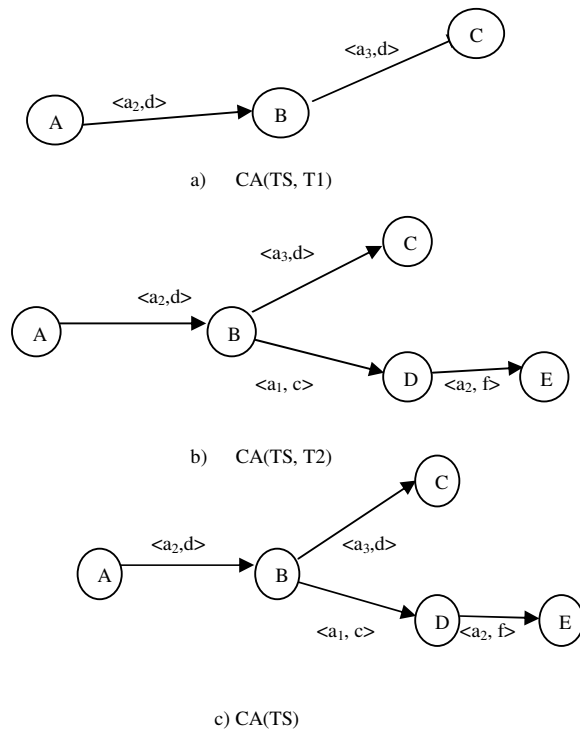


Figure 3. Two Constrained Attack Automata and the Target System Automaton

Consider again the automaton 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 $CA(TS, T_1)$, $CA(TS, T_2)$ and $CA(TS)$.

$CA(TS)$ describes the attacks that can be successfully implemented against TS as a function of the threats and of the elementary attacks. As far as concerns elementary attacks, it considers the resources each attack requires. Threats are described in terms of risk aversion and of the resources they can access. This is the minimal attack automaton that the assessment has to consider because no transition or no state may be removed from the automaton without losing information about the complex attacks that some threats can implement against TS .

4. RISK MITIGATION

After defining static and dynamic countermeasures, this section describes how constrained automata can model countermeasures and risk mitigation.

4.1 Attack Countermeasures

Given a constrained attack automaton, risk mitigation introduces a set of countermeasures for the attacks modeled by the automaton. Cost effectiveness may be achieved because the assessment considers only the attacks that at least one threat can, and is willing to, implement. We do not detail what a countermeasure is and define it as any security mechanism or policy that can prevent the success of an elementary attack against TS . It may consist of:

- a new version of a component that replaces the one with a vulnerability or applies a set of checks to prevent a threat from exploiting the vulnerability,
- a set of checks to discover an ongoing attack and some action to stop it.

A countermeasure is static if it is applied before attacks occur so that it can prevent the successful execution of any instance of the corresponding attack after its application. Instead, a dynamic countermeasure can be applied as the attack is going on only. In general, it updates the system configuration so that the attack is stopped before it can be completed.

Countermeasures introduce a further constraint on the elementary attacks that can be paired with the same label. As a matter of fact, since these attacks are modeled as instances of the same one, then there should exist at least one countermeasure that prevents all the attacks and that can be applied to any component affected by the vulnerability. If such a countermeasure does not exist, then the attacks cannot be considered as instances of the same one and the analysis should be repeated after

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

The return of a set of countermeasures is given by the impacts they avoid since some threats cannot achieve some of their goals. As any other component, a countermeasure may fail because of implementation error or faults in some components. Hence, redundancy can be introduced to further increase system robustness. In the following, we consider independent countermeasures only, i.e. we assume that a failure of one countermeasure does not result in the failure of another one.

4.2 Static Countermeasures

To model the adoption of static countermeasures, or defensive actions [4, 13, 24, 25], we remove from a constrained attack automaton the transitions paired with attacks prevented by the countermeasures. Consider an attack a_i associated with a transition of $CA(TS)$, if we know and apply a countermeasure for a_i , no threat can successfully execute a_i against c , hence $\langle s_1, s_2, a_i, c \rangle$ for any s_1 and s_2 , cannot belong to $CA(TS)$ after the countermeasure has been applied. Let $Acm(CA(TS))$ be the set of pairs $\langle a_i, c \rangle$ such that the countermeasure for a_i has been applied to c . We are interested in a complete set of countermeasures $Cocm(CA(TS))$, or in the critical set of attacks [4, 13, 24, 25, 35], that is in a set of pairs such that after removing the corresponding transitions, no sequence of transitions of

CA(TS) can reach a final state. A complete set of countermeasure is minimal if none of its subsets is complete. In the following, $Cocm(TS)$ denotes a complete set of countermeasure for the attack described by CA(TS) even if this neglects the set depends upon the automaton. To characterize $Cocm(TS)$, we consider $AG(TS)$, the labelled directed graph that describes the state transitions of CA(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
- an arc from $n(s_1)$ to $n(s_2)$ for each transition $\langle s_1, s_2, a, c \rangle$ of CA(TS). The arc is labeled by $\langle a, c \rangle$

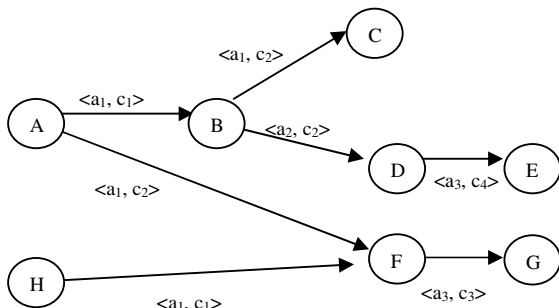


Figure 4. An Automaton CA(TS)

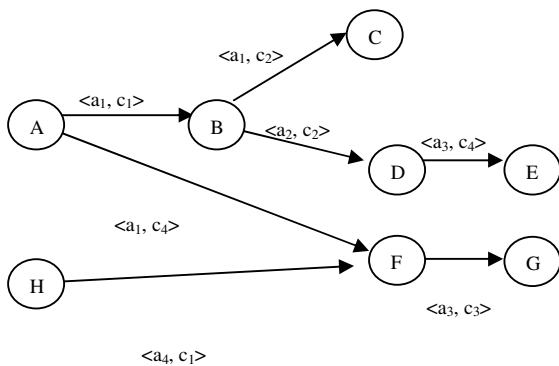


Figure 5. Complete and Minimal Sets of Countermeasures.

$AG(TS)$ is acyclic because $CA(TS)$ 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 final state can be reached and no countermeasure is required.

We recall that a set of arc $CS(G)$ of a graph G is a cut set of G if final node can be reached after removing all the arcs in S . 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 $AG(TS)$
- minimal iff $A(Cocm(TS))$ is a minimal cut set of $AG(TS)$.

Consider the graph in Fig.4 and assume that A and H are the initial nodes and C, E and G the final ones. The set including

the countermeasure for $\langle a_1, c_1 \rangle$ and that for $\langle a_1, c_2 \rangle$ is a complete one because by removing the corresponding arcs, no final state can be reached. The set is not minimal because the property holds even if we do not remove the arc $\langle a_1, c_2 \rangle$. Another complete, but not minimal, set is the one that includes the countermeasures for $\{ \langle a_2, c_3 \rangle, \langle a_3, c_4 \rangle, \langle a_1, c_2 \rangle \}$. The set of countermeasures for $\{ \langle a_1, c_2 \rangle, \langle a_3, c_4 \rangle, \langle a_3, c_3 \rangle \}$ 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 $\{ \langle a_3, c_3 \rangle$ and $\langle a_1, c_1 \rangle \}$.

The adoption of any countermeasure in a complete set prevents the successful execution of any complex attack because no final state can be reached after removing the arcs representing the corresponding elementary attacks. A set of countermeasures is minimal if one final state can be reached anytime at least one of its countermeasures is not applied. 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 **any** arc labeled $\langle la, c \rangle$. Only if the set of countermeasure is optimal, i.e. no arc is useless removed, the cut set is also a minimal one. In terms of the automaton, a set of countermeasures is complete if, for any final state and any path to the state, if include at least one transition along the path.

Another important notion strongly related to risk mitigation is that of **redundant set of countermeasures**. A redundant set include several countermeasures for the same elementary attack to take into account that a countermeasure could fail because of design or implementation errors or fault in a component. To handle this failure, independent countermeasures for the same attack can be adopted. A set of countermeasures is **k-redundant** if can prevent the success of any complex attack even if, at most, k of its countermeasures fail. As an example, a set of countermeasures is 2-redundant if it prevents the successful execution of any complex attack even if at most two of its countermeasures fail. Any set of countermeasures 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 countermeasures 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 prevent the threat from implementing any complex attack against TS .

More formally, a k -redundant set of countermeasures is the union of CM_1, \dots, CM_k , where for any $1 \leq i \neq j \leq k$

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

To prove this, consider that in the most general, and severe, case where all the arcs of the graph are labeled by a distinct pair $\langle a, c \rangle$ and no arcs on distinct paths from an initial state to a final one have the same label. A complete, and minimal, set of countermeasures CS can be built by considering one arc $\langle a, c \rangle$ for each path and by inserting into CS a countermeasure for $\langle a, c \rangle$. If an arc belongs to two minimal sets of countermeasure, a final state can be reached anytime

the corresponding countermeasure fails, hence the two sets do not define a k -redundant set for any $k \neq 0$. This implies that, in general, *each countermeasure that belongs to two sets of countermeasures reduces by one the degree of redundancy*. Furthermore, this also implies that a *k -redundant set can be defined only if each path from an initial state to a final one includes at least k arcs* because if the path is shorter, then all the countermeasures for the corresponding attacks may fail and the corresponding final state can be reached. Hence, path shorter than k prevents the definition of a k -redundant set because all the countermeasures for the elementary attacks that the path composes may fail.

4.2 Dynamic Countermeasures

Dynamic countermeasures are those countermeasures that do not remove the vulnerability but execute actions that prevent the transition of TS into a state where the threats achieve their goals. These countermeasures can be modeled as a set of actions to be executed to defend TS upon discovering that it has entered a state where a threat control some attributes she should not control, i.e. the security policy of TS forbids the threat from controlling the attributes. We assume that, on behalf of the system owner, a defender executes the actions in a state that implement the countermeasure to prevent an attacker, i.e. a threat, to reach a goal. As a consequence, elementary attacks fire some of the transitions of the automaton that models the attacks, while the defender actions fire some other transitions in some state. Obviously, to reach a goal a threat should fire a sequence of transitions ending in a final state. Instead the defender should fire a sequence of transitions that returns TS to an initial state or at least that prevents TS from reaching a final state. Some state can be paired with no action of the defender to models the case where the defender has no visibility of the state, i.e. the defenders cannot know that TS has entered into the state. Furthermore, only states that are not final can be paired with a defender.

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 consider the sequence of elementary attacks to be executed starting from an initial state, the equivalence relation among states and the defender actions for some of the classes. For each class, we also define whether the attacker can implement its attacks. If the attacker cannot implement its attack in the state of an equivalence class, then the defender action has to be defined for the class. The actions of the attacker or of the defender are defined a priori, independently of those of the opponent. At each step, we consider the current state of the automaton cs and ea , the next elementary attack that is the first action of the sequence of the attacker actions that still has to be considered. The following rule determines the next state of the automaton:

- 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 and the attacker action cannot be implemented, then the next state is determined according to ad ,
- if cs is paired with an action ad of the defender but the attacker action ad can be applied as well, 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. Otherwise, ea is not consumed and it can still be executed in the next states.

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

- any state in SE is a final one,
- any state in SE is an initial one,
- at least one state in SE is final,
- no set in SE is final and at least one is.

Case a) is complete success of the attack because the actions of the defenders are ineffective and only final states are reached. The reverse is true in case b) because a legal state of TS is restored, that is a state where each user and each threat controls the proper set of attributes only. 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. It may be considered as a success of the attacker. The last case is the most ambiguous one because TS is left in a state that is not legal and where new attacks can be more effective. It may be consider as a partial success of the defender.

Consider now an automaton execution ending 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. A state s belongs to the critical set of a final state if the choice of the action to be executed in s influences the final results.

In a further case, also the action of the attacker depends upon the 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 reached by the automaton, an empty action is possible
- distinct defender actions may be specified for each state. An empty action is paired with a state that is no visible for 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.

To automate the analysis, we can 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 [13]. Ideally, we could model each environment (attacker) that induces an outcome of its interactions on the system (defender) and check all the possible outcomes (attacks vs countermeasures).

5. CONCLUSIONS

We have presented a mathematical framework to support a formal approach to risk assessment. In particular, we have

considered attack automata that support the modeling of complex attacks as alternative sequences of elementary attacks. To determine the attacks that can be actually be executed, posets are introduced to evaluate the resources a threat can access and to compare them against those required to implement the attack. In this way, the automata that describe the complex attacks can be simplified by removing any elementary attack that no threat can implement. The automata also support the analysis of risk mitigation because the adoption of static countermeasures is formally described in terms of a cut set of the graph that describes a constrained attack automaton. Dynamic countermeasures are described by state transitions distinct from those that model elementary attacks.

An important open problem is the probability that an attack occurs and, hence, of the corresponding transition and the corresponding risk. A correct evaluation of this probability requires 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 Vulnerability: a Case Study Analysis, *IEEE Computer*, December 2002, p.52 - 59.
- [3] R. Baldwin, H.Kuang, *Rule Based Security Checking*, Tech. Rep., MIT Lab for Computer Science, May 1994.
- [4] F.Baiardi, C.Telmon, S.Suin, M.Pioli, Assessing the Risk of an Information Infrastructure through Security Dependency, Proc. of CRITIS '06, LCNS n. 4347, pp. 42-54, Nov. 2006.
- [5] M.Bishop, *Computer Security*, Addison Wesley, 2003.
- [6] CC-project, *Evaluation Methodology, Common Criteria for IT Security Evaluation*, CEM-99/045 Aug.1999.
- [7] CC-project, *User Guide. Common Criteria for IT Security Evaluation*, Oct. 1999.
- [8] 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
- [9] M.Dacier, *Towards Quantitative Evaluation of Computer Security*, Ph.D Thesis, Institute National Polytechnique de Toulouse, Dec 1994
- [10] 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.
- [11] C. W. Geib and R. P. Goldman, Plan Recognition in Intrusion Detection System, *DARPA Information Survivability Conference and Exposition (DISCEX II)*, June 2001.
- [12] R. P. Goldman, W. Heimerdinger, and S. A. Harp. Information Modeling for Intrusion Report Aggregation, *DARPA Information Survivability Conference and Exposition (DISCEXII)*, June 2001.
- [13] O. Kupferman and M.Y. Vardi, Module Checking, *8th Int. Conf. on Computer Aided Verification, LNCS 1102*, p. 75-86, 1997.
- [14] S. Jajodia, S. Noel, B. O'Berry, Topological Analysis of Network Attack Vulnerability, *Managing Cyber Threats: Issues, Approaches and Challenges*, Kluwer Academic Publisher, 2003.
- [15] 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.
- [16] U. Lindqvist, E. Jonsson, How to Systematically Classify Computer Security Intrusions, *1997 IEEE Symposium on Security and Privacy*, May 1997.
- [17] K. Lye, J. Wing, Game strategies in network security, *Foundations of Computer Security Workshop*, July 2002.
- [18] M. Howard, J. Pincus, J. M. Wing, Measuring Relative Attack Surfaces, *Proceedings of Workshop on Advanced Developments in Software and Systems Security*, Taipei, December 2003.
- [19] R. A. Martin, Managing vulnerabilities in networked system, *IEEE Computer*, Nov. 2001. p. 32 - 38.
- [20] F. Moberg, *Security Analysis of an Information System using an Attack Tree-based Methodology*, Master thesis, Chalmers University of Technology, 2000.
- [21] P. Moore, R. J. Ellison, R. C. Linger, *Attack Modeling for Information security and Survivability*, Technical note CMU/SEI-2001-TN001.
- [22] P. Ning, P.Cui, D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts", Proc. of the 9th ACM Conf. on Computer and Comm. Sec, November 2002, Washington, DC, USA.
- [23] 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.
- [24] P. Ning, D. Xu, Hypothesizing and Reasoning about Attacks Missed by IDSs, *ACM Trans. On Information System Security*, Vol.7, No.4, Nov. 2004, pp. 591-627
- [25] 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.
- [26] S. Jha, O. Sheyner, J. M. Wing. *Minimization and Reliability Analyses of Attack Graphs*. Technical Report CMUCS-02-109, Carnegie Mellon University, February 2002.
- [27] S. Jha, O. Sheyner, J. Wing, Two Formal Analysis of Attack Graphs, *15th IEEE Computer Security Foundations Workshop*, p.49, June 24-26, 2002.
- [28] 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.
- [29] X. Qin, W. Lee, Attack Plan Recognition and Prediction Using Causal Networks, *20th Annual Computer Security Applications Conference* pp. 370-379, 2004
- [30] 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.
- [31] B.Schneier, Attack Trees: Modeling Security Threats, *Dr. Dobbs's Journal*, December 1999.
- [32] 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.
- [33] O. M. Sheyner, *Scenario Graphs and Attack Graphs*, Ph.D. Thesis, CMU-CS-04-122, April 14, 2004.
- [34] D. Smith, J. Frank, A.Jonsson, Bridging the Gap Between Planning and Scheduling, *Knowledge Engineering Review*, 15(1), 2000.
- [35] L.P. Swiler, C. Phillips, D. Ellis, S. Chakerian, Computer-Attack Graph Generation Tool *DARPA Information Survivability Conference & Exposition*, June 2001.
- [36] F.Swideriski, W.Snyder, *Threat Modeling*, Microsoft Press, 2003.
- [37] S. J. Templeton, K. Levitt, A Requires/Provides Model for Computer Attacks, *Workshop on New Security paradigms*, p.31-38, September 2000,
- [38] S. Tidwell, R. Larson, K. Fitch, J. Hale, Modeling Internet Attacks, *IEEE Workshop of Information Assurance and Security*, June 2001

Author Biographies

F.Baiardi is a Full Professor with Dipartimento di Informatica, Università di Pisa where he chairs one of the computer science degree. His main research interest in the computer security field is risk assessment and management of complex ICT infrastructures.

Fabio Martinelli is a researcher with the Istituto di Informatica e Telematica, C.N.R., Pisa. His interests include formal analysis of network and system security, information flow analysis PKI and trust management, usage control. He teaches in graduate and postgraduate university courses.

Laura Ricci is an assistant professor with Dipartimento di Informatica, Università di Pisa. Her main research interests are distributed and P2P systems. In this fields, she is currently investigating P2P solutions for distributed information services and distributed virtual environments.

Claudio Telmon is a consultant on ICT security for small to multinational companies and for government agencies. His interests include risk assessment and audit of ICT systems and support in the development of Information Security Management Systems. He teaches in postgraduate university courses and in awareness seminars.