ASTRA : A Security Analysis Method Based on Asset Tracking

Daniel Le Métayer and Claire Loiseaux

Abstract ASTRA is a security analysis method based on the systematic collection and analysis of security relevant information to detect inconsistencies and assess residual risks. ASTRA can accommodate organizational as well as technical aspects of security and it can be applied to innovative products for which no security data (e.g. vulnerability or attack database) is available. In addition, ASTRA explicitly deals with the notion of responsibility and naturally leads to an iterative refinement approach. This paper provides an introduction to the method and comparison with related work.

1 Context and motivations

A broad variety of methods and techniques have been proposed for IT security analysis, both by the academic world and by industry, with a number of differences in terms of scope, objectives and approaches. Actually, even the perimeter of what is called "security analysis" and the meaning of the basic terms used in this area are subject to subtle variations [8]. In this paper, we refer to security analysis as a part of a more general "security management" (or "risk management") process, the goal of the security analysis being to prepare the technical arguments for a subsequent "decision making" phase, which typically involves business related considerations[1].

The context in which the ASTRA method has been devised is the delivery of security services for the design or certification of innovative IT products. Our ex-

Daniel Le Métayer
Inria Rhône-Alpes, 655 venue de l'Europe, Montbonnot e-mail: Daniel.Le-Metayer@inrialpes.fr · Claire Loiseaux
Trusted Labs, 5 rue du Bailliage, Versailles, e-mail: Claire.Loiseaux@trusted-labs.com

[1] Possible outcomes of the decision phase being to accept the risks, to mitigate them (e.g. through the implementation of additional countermeasures), to transfer them (e.g. to an insurance company) or to avoid them (e.g. by deciding not to distribute a new product or functionality).

perience is mostly with companies offering new solutions in the field of telecommunications, banking or e-administration. As far as technology is concerned, such products and services are usually based on smart cards, mobile phones and/or security modules. The needs of these companies in terms of security analysis typically occur at two stages of the life cycle of the new products: before the design phase, for example as part of a feasibility study, and during (or even after) development, to prepare a subsequent certification procedure.

The first qualities of a security analysis method in this context are rigour, generality and incrementality:

- *Rigour* is obviously a virtue of any method, but it is a prerequisite for any method to be used in a certification process (especially if the highest levels of assurance are to be targeted). Rigour can be achieved through the application of systematic rules. Systematization itself brings additional benefits: it improves the efficiency of the process (and therefore reduces delays and costs, which are crucial factors in the case of innovative products for which time-to-market is often decisive) and enhances repeatability and maintenance.
- *Generality* is the ability to cope with all security aspects, including organizational, technical as well as management issues. The lack of generality, or the inability to provide a complete view of all security aspects, may lead to overlook significant issues or to spend too much energy and time on minor items when other, more significant, aspects, are underestimated.
- *Incrementality* is also crucial because, in practice, security analyses can rarely be one shot undertakings. Most companies prefer to start with a preliminary analysis, which should produce first conclusions as soon as possible at a moderate cost, before deciding to embark on more extensive studies.

From our experience, one of the main challenges for the security analyst is to be able to provide a representation of security which is both sufficiently complete and sufficiently rigorous. Actually, rigour is necessary at two levels:

- At the descriptive level: in most cases (and not only in large organizations), security information is spread over different groups of actors (architects, developers, suppliers, managers, security experts, etc.). One of the main tasks of any security analysis is therefore to gather all relevant information and build an overall picture of the security of the system. Needless to say, one usually observes different views among different actors. In any complex system, inconsistencies may also arise within individual representations of the system[2]. Such inconsistencies are typical symptoms (if not the sources) of misconceptions and vulnerabilities: detecting them is thus the first major outcome of a security analysis, which is possible only through a rigorous approach.
- At the analysis level: one of the main goals of a security analysis is to provide technical arguments for further decisions concerning the system (e.g. enhancement, deployment, security certification level, etc.). The key issue to this respect

[2] Typically, different assumptions can be made about the security features or available functionalities of a component at different design stages.

is to be able to justify such decisions: to this aim, the results of the security analysis should come with sound rationales and tracing facilities. Traceability and precise rationale, which are required by certification procedures such as the Common Criteria [4], also facilitate the maintenance of the security of the system.

Last but not least, a sufficient level of rigour is also necessary in order to establish the precise responsibilities of all actors and stakeholders. Responsibility can be understood here both in the technical sense and the legal sense (liability). Indeed, a large number of actors are usually involved in the design and operation of modern IT products and systems[3] and security issues may increasingly become a matter of liability, especially when substantial valuables are at stake.

Evaluating existing security analysis methods by the above yardsticks leads us to their classification into two main categories:

- In the first category, which includes most industrial methods and standards [1, 11, 15, 16], some level of systematization is attained through the use of catalogues or checklists. Checklists are a very effective way to capitalize on past experience and reduce the dependency of an organization with respect to a small group of experts. However, apart from systematization, they do not introduce by themselves a high level of rigour. In addition, they are appropriate only for the analysis of established (and relatively stable) categories of products such as operating systems or firewalls: they cannot be applied to the analysis of new products in emerging markets for which, typically, no data base of vulnerabilities is yet available.
- Methods in the second category provide a systematic approach based on semi-formal or formal models of the system under study [2, 3, 7, 12, 13]. Different levels of rigour can be attained depending on the formalism used to represent the models and the tools available to analyse them. However theses methods, which originate mostly from the academic world, usually focus on technical issues and leave organizational aspects out of their scope.

The ASTRA method has been devised precisely to fill this gap and provide a framework for the systematic security analysis of innovative products, addressing in an incremental and uniform way both organizational and technical aspects. The method is iterative and relies on the systematic collection and analysis of all security relevant information to detect inconsistencies and assess residual risks. In this paper, we present an introduction to the method and relate it with previous work. The framework is introduced in Section 2, followed by a presentation of the method itself in Section 3. Section 4 discusses related work and Section 5 summarizes the benefits and limitations of the method.

[3] For a device as small as a mobile phone, one can think of the device provider, operating systems and software suppliers, content providers, the operator, not to mention the user himself who plays an increasing role in the management of his device.

2 The Framework

The core of the ASTRA method is the construction and analysis of functions representing different views of the system (*Security Views*). Before entering into the presentation of the method itself in Section 3, we first provide the basic notions in Subsection 2.1 and introduce the three main components of the *Security Views* in the following subsections: right functions in Subsection 2.2, responsibility functions in Subsection 2.3 and dependency relations in Subsection 2.4.

2.1 Basic Notions

As its name suggests, ASTRA is based on the idea of *asset tracking*. In addition to assets, the basic ingredients of the method are locations, subjects, access rights, contexts, trust levels and sensitivity levels:

- An *asset* can be anything (part of the IT product or under its control) which has a value and needs to be protected. Assets can be digital (e.g. health record, cryptographic key, PIN, etc.) or physical (e.g. USB key, computer, network, official authorization letter, etc.). In the following, A denotes the set of assets.
- In order to track assets, we use the notion of *location*: a location can be seen as a container for assets; in other words, access to assets is possible only through locations. Locations can take different forms: computer memory, compact disk, network (cable or wireless), office cabinet, computer room, etc. The set of locations is denoted by L.
- As usual, *subjects* are the active entities in the system: subjects can have access to assets and locations. Again subjects can be digital (software code) or physical (developer, security officer, night-watchman, etc.). Note that the notion of subject used here is different from the notion of *legal entity* which can be considered in an extension of the method as set forth in Section 5. U denotes the set of subjects.
- Subjects may have *access rights* to assets and to locations. At first glance it could seem that considering both types of access rights just introduces useless redundancies. We believe that there are good reasons to include both of them though: first, they do not convey exactly the same kind of information and, from our experience, some actors feel it more natural to reason in terms of assets and others in terms of locations; also, one of the goals of security analysis is precisely to detect inconsistencies: in this context, offering the possibility to introduce redundancies is thus an advantage rather than a weakness.
- Access rights may depend of the current *context*. The context is a property of the state of the system, it being understood that states can encompass technical as well as procedural information (e.g. execution mode, security status, presence of a security officer, official authorization letter, etc.). For the sake of uniformity, we consider that the state Δ is a set of designated assets.

- Each subject s is associated with a *trust level T(s)* and each asset a is associated with a *sensitivity level S(a)*. Trust and sensitivity levels play an instrumental role in the evaluation of risk levels. They should thus be chosen with great care by the security analyst. To remedy any potential misjudgement in their assessment, the ASTRA method makes it easy to play *what-if games*, typically by making different assumptions about trust levels and analysing their consequences in terms of risks. In addition, as further detailed below, trust levels can be used to place different levels of constraints (whether technical, organizational or legal) on subjects.

2.2 Right functions

The three main functions which form the core of the ASTRA method are the Subject-Location function, the Subject-Asset function and the Asset-Location function:

- The Subject-Location function $SL_r(s,l)$ defines, for each subject s and each location l, the context c in which subject s has access right r to location l. More precisely, s may not have access right r to l, except when $SL_r(s,l)$ holds. The access right r can be *read* or *write*.
- The Subject-Asset function $SA_r(s,a)$ defines, for each subject s and each asset a, the context c in which subject s has access right r to asset a. More precisely, s may not have any access to a, except when $SA_r(s,a)$ holds. The semantics of *read* for assets is the possibility to obtain information about a while *write* includes the modification, creation, deletion and copy of the asset (all operations modifying the set of values of the asset in the system).
- The Asset-Location function $AL(a,l)$ defines, for each asset a and each location l, the context c in which location l may contain information about asset a. More precisely, l may not contain any information about a, except when $AL(a,l)$ holds.

2.3 Responsibility Functions

As set forth in the introduction, we believe that responsibilities should be dealt with explicitly in a security analysis method. Responsibilities can be specified through the following E (for *Ensures*) functions in ASTRA:

- $E(SL_r)(s,l) = \{s'\}$ specifies that subject s' is responsible for ensuring that subject s has access to location l only in context $SL_r(s,l)$. Obviously, we may have $s = s'$, which means that the subject is responsible for its own access to l, but it is not necessarily the case (and it should not be for subjects with low trust levels).
- $E(SA_r)(s,a) = \{s'\}$ specifies that subject s' is responsible for ensuring that subject s has access to asset a only in context $SA_r(s,a)$. In contrast with $E(SL_r)$,

which must return a non empty set of subjects, we may have $E(SA_r)(s,a) = \emptyset$, which means that no subject is explicitly designated as responsible for this rule. Instead, the rule has to be ensured indirectly, through conditions imposed by the SL_r and AL functions. Such under-specifications are often used for Subject-Asset relations because their implementation can be indirect: typically, a subject may have no access to a given asset because it has no access to a location which may contain this asset. Note that this does not mean that such rule will be left without any responsible subject: as shown in Section 3, responsibilities will instead be derived by the method. In addition, we impose that $\forall a \in \Delta, E(SA_{write})(s,a) \neq \emptyset$: in other words, the subjects responsible for context changes must be explicitly identified. This constraint is both reasonable, because of the instrumental role played by contexts, and useful from a technical point of view (see Subsection 3.2).

- $E(AL)(a,l) = \{s\}$ specifies that subject s is responsible for ensuring that information about asset a can be found in location l only in context $AL(a,l)$. For the same reason as $E(SA_r), E(AL)$ can return $\emptyset$, which means that the responsibilities for the corresponding rule will be derived by the method.

To conclude this subsection, let us note that, for the sake of generality, responsibility functions return sets of subjects. In most cases however, it is advised that this set should be reduced to a singleton.

2.4 Dependency Relations

For better clarity and conciseness it is useful in practice to define dependencies between assets and between locations: for example a message asset may be made of several fields, each of them containing another asset; an asset (e.g. ciphered text) can be derived from other assets (e.g. clear text and cryptographic key); a location can be a memory zone containing several buffers, each of them considered as another location. To address this need, we consider simple dependency relations between locations and assets respectively:

- $l \subseteq l'$ specifies that location l is included into location l'.
- $a_1, \ldots, a_n \rightarrow a$ means that the knowledge of information about assets $a_1, \ldots$ and a_n may provide information about asset a.

The dependency relations are implicitly closed by transitivity. Note that $a_1, \ldots, a_n \rightarrow a$ does not necessarily entails $a \rightarrow a_i$ for any a_i because a may be obtained from $a_1, \ldots, a_n$ through a one-way function. More sophisticated dependencies can be considered but we found the above relations sufficient in practice.

3 The Method

In the following, we introduce successively the two main phases of the method: the collection of information and detection of inconsistencies in Subsection 3.1 and the risk assessment in Subsection 3.2. Both phases are iterative. The first one constitutes the initial phase of the analysis, whose goal is to build a consistent and comprehensive view of the security of the system. The second one is repeated, possibly with intermediate decision making steps (e.g. to decide the implementation of additional countermeasures) until a stable state is reached. The two phases identify different kinds of pathological situations: sheer contradictions for the first phase and high risk levels for the second one.

3.1 Collection of Information and Detection of Inconsistencies

Before starting the collection of information, the very first task of any security analysis should be to precisely fix the perimeter of the analysis and assumptions about the actors involved. In ASTRA, the perimeter is defined by the sets of assets, locations and subjects, with the associated assumptions about levels of trust and levels of sensitivity and the dependency relations. This task is crucial because it defines the objectives and limitations of the analysis. The assessment of trust and sensitivity levels is especially delicate and should take account of technical as well as business considerations. This issue is further analysed in Subsection 3.2 where trust and sensitivity levels are used to derive risk levels.

As set forth in the introduction, various pieces of information about the security of a system are usually spread over several groups of actors. In addition, different actors may have different views about the functionalities and the security of the system. Such differences sometimes reveal serious misconceptions about the system which may lead to major security holes. To tackle this issue, the first task in ASTRA is to consolidate all relevant information in the form of *Security Views*: each *Security View* consists of SL_r, SA_r, AL and E functions representing the view of one actor or group of actors (e.g. requirements team, security expert, architect, developer team, integrator, etc.). Two types of inconsistencies may occur in *Security Views*, inconsistencies between different views and inconsistencies within a single view, which are defined in the following subsections.

3.1.1 Inter-view inconsistencies

Inter-views inconsistencies occur when $F_1(x,y) \neq F_2(x,y)$ where F_1 and F_2 stand for the versions of one of the SL_r, SA_r, AL and E functions in *Security Views* 1 and 2 respectively. From our experience, the two most common cases of inter-view inconsistencies are the following:

- For SL_r, SA_r and AL functions, one of the two conditions (or state properties) may be strictly weaker than the other one, that is $F_1(x,y) \Rightarrow F_2(x,y)$ or $F_2(x,y) \Rightarrow F_1(x,y)$, which means that one category of actors has placed stronger security requirements on a component than the other one.
- For E functions, $F_1(x,y) \neq F_2(x,y)$ reveals different assumptions about the subjects responsible for ensuring a security rule.

If not simple oversights, both kinds of inconsistencies may be the symptoms of serious discrepancies about security issues among different teams and have to be solved through discussions between these teams. As a result, one *Security View* at least has to be modified. The impact is sometimes confined to the presentation of the requirements or share of responsibilities but it may also happen that the implementation itself is affected.

3.1.2 Intra-view inconsistencies

We distinguish two kinds of intra-views inconsistencies: right inconsistencies and responsibility inconsistencies, which are defined in Definitions 1 and 2 respectively.

Definition 1. A *right inconsistency* occurs within a view if one of the following conditions holds:
(1) $l \subseteq l'$ *and* $\neg(SL_r(s,l) \Rightarrow SL_r(s,l'))$
(2) $a_1,\ldots,a_n \rightarrow a$ *and* $\neg(SA_r(s,a_1) \wedge \ldots \wedge SA_r(s,a_n) \Rightarrow SA_r(s,a))$
(3) $l \subseteq l'$ *and* $\neg(AL(a,l) \Rightarrow AL(a,l'))$
(4) $a_1,\ldots,a_n \rightarrow a$ *and* $\neg(AL(a_1,l) \wedge \ldots \wedge AL(a_n,l) \Rightarrow AL(a,l))$

Definition 2. A *responsibility inconsistency* occurs within a view if the following two conditions holds:
(5) $E(SA_r)(s,a) = \emptyset$
(6) $\exists l \in L, \neg(AL(a,l) \wedge SL_r(s,l) \Rightarrow SA_r(s,a))$

Right inconsistencies occur when the rights defined through the SL_r, SA_r and AL functions are in contradiction with the dependencies between locations or assets. A contradiction derives from (1) in Definition 1 in a situation where, for example, a memory zone l' contains a block l (thus $l \subseteq l'$), subject s has unconditional access rights to l (thus $SL_r(s,l) = True$), but no access right on l' (thus $SL_r(s,l') = False$). Note that the opposite condition is not imposed, which means that access rights to a location can be restricted to certain subsets of this location[4].

The second condition in Definition 1 identifies situations where access is allowed to intermediate assets $a_1,\ldots,a_n$ which, together, can be used to get information about an asset a which should not be accessible.

Condition (3) mirrors Condition (1) for Asset-Location rights: if a memory zone l' contains a block l (thus $l \subseteq l'$), which can contain information about an asset a

[4] This is consistent with the semantics of $SL_r(s,l)$ presented above : subject s may not have any access right r to l, except when $SL_r(s,l)$ holds

(thus $AL(a,l) = True$), then l' should also be allowed to contain information about a (thus $AL(a,l') = True$).

Finally, Condition (4) mirrors Condition (2): if location l is allowed to contain intermediate assets $a_1, \ldots, a_n$ which, together, can be used to get information about an asset a, then l should be allowed to contain information about asset a.

Right inconsistencies thus stem from overlooked dependencies: sometimes they can be corrected without deep impact on the design of the product (e.g. through the extension of unduly restricted access rights in the *Security View*) but there are also cases where they call for more drastic modifications to some components or even to the architecture of the product (e.g. when the current design does not allow for sufficient protection of intermediate assets).

A responsibility inconsistency occurs when the *Security View* does not specify any responsible subject for a Subject-Asset access rule and does no provide any way to ensure this access rule indirectly. As set forth in Subsection 2.3, $E(SA_r)(s,a) = \emptyset$ means that the condition $SA_r(s,a) = c$ is to be ensured indirectly, through conditions imposed by SL_r and AL. The property to be checked is thus $\forall l \in L, SL_r(s,l) \wedge AL(a,l) \Rightarrow SA_r(s,a)$, which expresses the fact that if subject s can get access to a location l which may contain asset a, then such access is permitted only in the context specified by $SA_r(s,a)$. If this property is not satisfied, a contradiction occurs in the *Security View* because the responsibility to ensure $SA_r(s,a)$ cannot be assigned to any subject.

Information collection and detection of inconsistencies is an iterative and interactive process: when inconsistencies are discovered, either intra-view or inter-view, discussions are organized with the concerned teams to further study the conflicting rules and strengthen the overall understanding of security issues. In the most serious cases, it may even be necessary to set up global project meetings to converge towards a common view. From our experience, this first phase goes much further than a simple collection of information: the elucidation of the expectations and assumptions of the different actors already makes it possible to uncover major gaps or misconceptions about security issues.

3.2 Risk Assessment

The result of the first phase is a single consolidated *Security View* consisting of SL_r, SA_r, AL and E functions. Trust and sensitivity levels are also attached to subjects and assets during this first phase. The second phase consists in identifying and classifying risks based on the consistent set of information resulting from the first phase. In contrast with the inconsistencies identified in the first phase, risks are not sheer contradictions: they represent potentialities of attacks which can be classified on a severity scale.

In ASTRA, risks are identified and assessed based on a ternary relationship between the rights, the subject responsible for those rights and the assets at stake. More precisely, for each right specified by a function F in SL_r, SA_r or AL, we derive:

- The set $C(F)$ of assets concerned by the right and
- The set $R(F)$ of subjects responsible for enforcing the right.

The risk level associated with the right is then derived from the sensitivity level of the most sensitive asset in $C(F)$ and the trust level of the less trustable subject in $R(F)$. Different scales can be used to express sensitivity and trust levels, depending on the complexity of the system under study and the types of actors involved (authorized as well as malicious). Similarly, different algorithms can be used to derive risk levels. For the sake of illustration, we use very simple scales and algorithm here, that we found sufficient in most situations:

- Trust levels take integer values in a range from 1 to 4 (4 corresponding to the less trustable subjects).
- Sensitivity levels take integer values in a range from 1 to 4 (4 corresponding to the most sensitive values).
- The risk level associated with a right specified by F is the sum of the sensitivity level of the most sensitive asset in $C(F)$ and trust level of the less trustable subject in $R(F)$.

A risk level of 8 corresponds to the worst situation (the less trustable subjects being responsible for the security of the most sensitive assets), 7 corresponds to an unacceptable risk and 6 to a significant risk. Risks from level 5 to 2 are considered as "tolerable" (5 being the minimum level of risk for assets of sensitivity 4). Obviously, this interpretation depends very much on the interpretation of the trust and sensitivity levels themselves, and can be adapted to fit the needs of each project.

We proceed now with the definition of the risk levels, which are computed from the sets of responsible subjects and concerned assets.

Definition 3. The sets of responsible subjects R, the set of assets concerned C, and the derived risk level *Risk* are defined as follows for, espectively, SL_r, SA_r and AL:

(1) $R(SL_r)(s,l) = E(SL_r)(s,l)$
(2) $C(SL_r)(s,l) = D(\{a \mid AL(a,l) \neq False\})$
(3) $Risk(SL_r)(s,l) =$
$\quad Max(\{T(s') \mid s' \in R(SL_r)(s,l)\}) +$
$\quad Max(\{S(a) \mid a \in C(SL_r)(s,l)\})$
(4) $R(SA_r)(s,a) =$
$\quad if\ E(SA_r)(s,a) \neq \emptyset$
$\quad then\ E(SA_r)(s,a)$
$\quad else \bigcup \{R(SL_r)(s,l) \mid l \in L\} \bigcup \{R(AL)(a,l) \mid l \in L, SL_r(s,l) \neq False\}$
(5) $C(SA_r)(s,a) = D(\{a\})$
(6) $Risk(SA_r)(s,a) =$
$\quad Max(\{T(s') \mid s' \in R(SA_r)(s,a)\}) +$
$\quad Max(\{S(a) \mid a \in C(SA_r)(s,a)\})$
(7) $R(AL)(a,l) =$
$\quad if\ E(AL)(a,l) \neq \emptyset$
$\quad then\ E(AL)(a,l)$

$else \bigcup \{R(SL_r)(s,l) \mid s \in U\} \bigcup \{E(SA_{write})(s,a') \mid s \in U, a' \in \Delta\}$

(8) $C(AL)(a,l) = D(\{a\})$

(9) $Risk(AL)(a,l) =$
$Max(\{T(s') \mid s' \in R(AL)(a,l)\}) +$
$Max(\{S(a) \mid a \in C(AL)(a,l)\})$

Definition 4. $D(A)$ is defined as the minimal superset of A such that
$\forall a_1 \in D(A), \ldots, \forall a_n \in D(A),\ a_1, \ldots, a_n \rightarrow a \Rightarrow a \in D(A)$

Generally speaking, when specified in the *Security View* through the E function, the responsible subject is as defined by E ((1) and (4) in Definition 3).

When no responsible subject is specified for a Subject-Asset right $SA_r(s,a)$, then a subject is considered responsible for this right either if ((4) in Definition 3) it is responsible for the access right to a location by s (and thus it belongs to $R(SL_r)(s,l)$) or if it is responsible for the right for a location to contain a (and thus it belongs to $R(AL)(a,l)$). The motivation is that, as set forth in Subsection 3.1.2, the Subject-Asset right is ensured indirectly in such case, through Subject-Location and Location-Asset rights.

As far as Asset-Location rights are concerned, $AL(a,l)$ is an invariant property which can be breached by two kinds of actions: accesses to location l resulting in information about asset a being available in location l in an unauthorized context or changes of context. Responsible subjects are thus members of $R(SL_r)(s,l)$ and $E(SA_{write})(s,a')$, as set forth in (7) above[5].

The set of concerned assets is straightforward in the cases of $SA_r(s,a)$ and $AL(a,l)$ which apply directly to asset a ((5) and (8) in Definition 3). Note however that asset dependencies are taken into account through function D: if assets $a_1, \ldots, a_n$ are concerned by a rule and information about asset a can be derived from $a_1, \ldots, a_n$ then a is also concerned (Definition 4). The assets concerned by $SL_r(s,l)$ are all the assets which can reside in location l ((2) in Definition 3). Finally risk levels are defined for each function as set forth above, based on the trust levels of responsible subjects and the sensitivity levels of the concerned assets (lines (3), (6) and (9) in Definition 3).

To conclude this subsection on risk assessment, let us stress the fact that the above definitions allow us to separate the issues of defining the set of responsible subjects and evaluating of the risk level. Whereas the risk level depends on the initial assumptions about trust and sensitivity of subjects and assets, the definition of responsible subjects does not rely on such assessments. An interesting property of the definition of R above is that it leads to a confinement of responsibilities which can stated as follows:

Confinement Property:

1. If all the subjects in $R(SL_r)(s,l)$ comply with their responsibilities, as defined by R, then $SL_r(s,l)$ will be guaranteed by the system.

[5] Note that, as set forth in Subsection 2.3, $\forall a' \in \Delta, E(SA_{write})(s,a') \neq \emptyset$.

2. If all the subjects in $R(SA_r)(s,a)$ comply with their responsibilities, as defined by R, then $SA_r(s,a)$ will be guaranteed by the system.
3. If all the subjects in $R(AL)(a,l)$ comply with their responsibilities, as defined by R, then $AL(a,l)$ will be guaranteed by the system.

The proof of this property, which is omitted here for space considerations, is based on a case analysis of Definitions 3 and 4. The significance of this confinement property is that it holds disregarding the behaviour of the other subjects, that is to say even if they do not comply with their own responsibilities. This confinement property shows that Definitions 3 and 4 form a sound basis for risk assessment, independently of the validity of the assumptions about trust and sensitivity levels. Actually different assumptions about these levels can be made during the analysis to study their impact on the resulting risk based on the above definition of R.

4 Related Work

The security analysis methods used in industry fall essentially into two categories [10]: commercial methods and standards. As far as standards are concerned, [5] is essentially a code of good practices: it offers guidelines and very general principles, with strong emphasis on management and organizational issues while [4] as an international standard for the evaluation of IT products. [1] puts forward a general three phases approach based on (1) the establishment of an asset-based threat profile, (2) the identification infrastructure vulnerabilities and (3) the development of security strategy and plans. [15] is also very generic but more technical than [1]; it puts emphasis on the integration of risk management into the software development life cycle and proposes a decomposition of the analysis into a series of predetermined phases split into a number of steps with specified inputs, outputs and guidance (checklists, definitions, questionnaires, interview outline, etc.).

Turning to commercial methods, [11] is consistent with the recommendations in [15]; its main emphasis is cost-effectiveness and it provides organizational rules for conducting the analysis process based on brain storming meetings (people involved, roles, responsibilities, required material, meeting preparation, duration, objectives, checklists, etc.). The methods put forward by Microsoft [16, 6] are based on a range of information representation patterns (tables), classification lists (e.g. the STRIDE checklist for threats: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) and semi-formal representations (data flow diagrams, attack trees).

Attack trees [14] are a natural way to represent security attacks which is used by several methods. The root of an attack tree represents the goal of the attack and the nodes correspond to subgoals linked by *AND* and *OR* relations. Most interestingly, different kinds of attributes can be assigned to the leaves (cost, risk for the attacker, required equipment or expertise, etc.) and propagated to the root. Attack trees are rather popular in industry because they can be used in a pragmatic, incremental process in which leaves requiring deeper investigation can be progressively

decomposed into subtrees. Attack trees have limitations though; in particular, they provide a convenient framework for presenting, categorizing and analyzing attacks but they offer little help for the discovery of these attacks, which still relies on the expertise of the analyst.

Attack trees were originally presented in an informal way. Two main approaches have been followed to provide a formal framework for attack trees:

- The first approach consists in enhancing the attack tree notation with statements expressed in an attack specification language: the attack tree notation is then used as a structuring framework or glue syntax for the formal statements [17].
- The second approach is to endow the attack tree notation itself with a mathematical semantics, which makes it possible to study attack trees as mathematical objects of their own [9]. This approach can be used to define a class of "reasonable attributes" which can correctly be propagated bottom-up.

The attack tree approach has also been extended to *attack graphs* and techniques have been proposed for the automatic generation of attack graphs (based on attack templates) and their analysis (based on graph algorithms) [2, 7, 12], which is especially useful for the network security analysis.

A more detailed introduction to existing security analysis methods can be found in [8]. This quick review reveals several gaps in the security analysis landscape: first, most methods fall into one of the following two categories: they are either (1) general purpose with, usually, much emphasis on organizational issues or (2) based on a formalism with, usually, strong emphasis on systematization (or even automation) and rigour. As set forth in the introduction, attaining high levels of both generality and rigour (or systematization) remains a challenge. Attack trees and attack graphs go in the right direction but, as mentioned above, they offer little help to discover attacks in the first place. It remains necessary to rely on the expertise of the analyst or on the existence of catalogues or checklists which are usually not available for new, innovative products.

5 Conclusion

The method presented in this paper was devised precisely to fill the gaps identified in the previous section. The framework provided by ASTRA is:

- Rigorous : it can serve as a basis for inconsistency detection.
- General: it can accommodate organizational as well as technical aspects of security.
- Suited to innovative products for which no security data (e.g. vulnerability or attack database) is available.

In addition, the method naturally leads to an iterative refinement approach: inconsistencies or high risk levels can be solved through different kinds of refinements: decomposition of a subject (or code) into several subjects (for a more appropriate

allocation of responsibilities), decomposition of a context into more precise state conditions (to refine access rights), decomposition of a location into different sublocations (which should be associated with different access rights).

An important aspect of ASTRA is that organizational rules can be handled in exactly the same way as technical rules: individual actors such as security officers or a night-watchers can be represented as subjects, physical goods or authorization documents can be represented as assets, rooms or premises are represented as locations, etc. Actually malicious actors can also be included in the *Security Views* (with an extended trust level range to take into account the very low, or negative, level of trust associated with such subjects) and possibilities such as monitoring the electrical activity of a device to perform a DPA (Differential Power Analysis) attack can be represented as a form of location access (access to the room and access to the device, possibly in specific contexts such as the absence of security officer and night-watcher, etc.). Another possibility of the framework which has not been presented here is the use of sensitivity levels depending on rights: for example, for some assets *read* is a more sensitive type of access than *write*, for others the opposite holds.

Among the benefits of the method let us also stress the significance of the notion of responsibility: in practice, it turns out to be very useful to be able to separate the specification of the behaviour of a subject from the definition of the responsibilities for this behaviour. Indeed, many security holes in a system come from misunderstandings or conflicting views about responsibilities. The next step in this direction will be the introduction of legal entities associated with subjects, which will allow us to encompass legal liability.

As far as limitations are concerned, the method, in its current state, is targeted towards the study of invariant security properties such as confidentiality and integrity. More work is needed to extend it to denial of service or liveness properties (such as agreement properties). Last but not least, the application of a method like ASTRA should obviously be supported by a tool because the amount of available information is usually so large that its analysis by human beings would be cumbersome and error prone. The two main features of such a tool are its interface and its rule based engine. The role of the engine is to implement the rules set forth in Section 3 and to sort access rules according to associated the risk levels. The most challenging part of the tool is actually the user interface, which should be easy to use both by analysts (to enter security information, e.g. through dedicated text processing and spreadsheet functionalities and to trace the risk level calculation in order to understand how to improve the situation) and by decision makers (with simplified presentations of the results and alternative options). Dedicated graphical features are necessary, especially to support "what-if games" (changing assumptions about trust and studying the consequences in terms of risk levels) and to provide user-friendly representations of access rights.

Acknowledgements This work has been partially funded by the ANR (Agence Nationale de la Recherche) under the grant ANR-07-SESU-007 (project LISE: Liability Issues in Software Engineering).

References

1. Alberts, C., Dorofee, A., Stevens, J., Woody, C.: Introduction to the OCTAVE approach. Carnegie Mellon, SEI (2003).
2. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. Proceedings of the 9th ACM conference on Computer and Communications Security CCS'02(2002).
3. Besson, F., Jensen, T., Le Métayer, D., Thorn, T.: Model checking security properties of control flow graphs. Journal of Computer Security, Vol. 9 (2001).
4. Common Criteria for Information Technology Security evaluation. http://www.commoncriteriaportal.org/, CC V3.1 (2006).
5. ISO/IEC 17799:2005, Information technology - security techniques - code of practice for information security management (2005).
6. Howard, M., LeBlanc, D.: Writing secure code. Microsoft Press (2003).
7. Jha, S., Sheyner, O., Wing, J.: Two formal analyses of attack graphs. Proceedings of the 15th Computer Security Foundations Workshop, IEEE Computer Society (2002).
8. Le Métayer, D.: IT Security analysis: best practices and formal approaches. Proceedings of FOSAD Summer School (Formal Security Analysis and Design), Springer Verlag LNCS 4677 (2007).
9. Maw, S., Oostdijk, M.: Foundations of attack trees. International Conference on Information Security and Cryptology, Springer Verlag LNCS 3935 (2005).
10. McGraw, G.: Software security: building security in. Addison Wesley Professional (2006).
11. Peltier, T. R.: Information Security Risk Analysis. Auerbach Publications (2005).
12. Phillips, C., Swiler, L. P.: A graph-based system for network-vulnerability analysis. Proceedings of the 1998 Workshop on New Security Paradigms, ACM Press (1998).
13. Ramakrishan, C. R., Sekar, R.: Model-based vulnerability analysis of computer systems, Second International Workshop on Verification, Model Checking and Abstract Interpretation, (VMCAI'98) (1998).
14. Schneier, B.: Attack trees, modeling security threats. Dr Dobbs Journal (1999).
15. Stoneburner, G., Goguen, A., Feringa, A.: Risk management guide for information technology systems. NIST Special Publication 800-30 (2002).
16. Swiderski, F., Snyder, W.: Threat modeling. Microsoft Press (2004).
17. Tidwell, T., Larson, R., Fitch, K., Hale, J.: Modeling internet attacks. Proceedings of the 2001 IEEE Workshop on Information Assurance and Security (2001).