# From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering

Axel van Lamsweerde, Simon Brohez, Renaud De Landtsheer and David Janssens
*Département d'Ingénierie Informatique*
*Université catholique de Louvain*
*B-1348 Louvain-la-Neuve (Belgium)*
*{avl, sbr, rdl, dja}@info.ucl.ac.be*

**Abstract** *Caring for security at requirements engineering time is a message that has finally received some attention recently. However, it is not yet very clear how to achieve this systematically through the various stages of the requirements engineering process. We briefly introduce some of the requirements such a process should meet for high assurance to be provided from the resulting requirements product. A constructive approach to security requirements elicitation, modeling and analysis is then outlined as an attempt to address such meta-requirements. The approach is based on a framework we developed before for generating and resolving obstacles to requirements achievement. Our framework integrates intentional obstacles (or "anti-goals") set up by attackers to break security goals. Attack trees are derived systematically through anti-goal refinement until leaf nodes are reached that are software vulnerabilities observable by the attacker or anti-requirements implementable by this attacker. New security requirements are derived by resolution of the attack trees generated thereby.*

## 1. Introduction

Security has become an increasingly growing concern at the internet age. The number of security incidents reported to CERT has been growing exponentially over the past decade – from about 250 in 1990 to 2,500 in 1995 to 10,000 in 1999 to 22,000 in 2000 to more than 50,000 in 2001 and 80,000 in 2002 [CERT]. Software applications are more and more ubiquitous, heterogeneous, mission-critical and vulnerable [Kem03]; attackers are more and more malicious and use more and more sophisticated attack technology; the consequences of attacks may become more devastating up to the point of breaking severe safety-critical concerns. For example, there has been reports about denial of service on medical records that prevented urgent surgery from being undertaken under the right conditions [CERT].

Unsurprisingly, the major source of vulnerability has been recognized to be poor-quality software [Vie01]. As far as research is concerned, the state of the art has been fairly unbalanced so far. As Wing noted, the "strength" of security guarantee provided by current security technology is inversely proportional to the "size" of the software layer at which the technology applies [Win98]. Such contrast between the size of the problem space and the size of the solution space is summarized in Fig. 1.

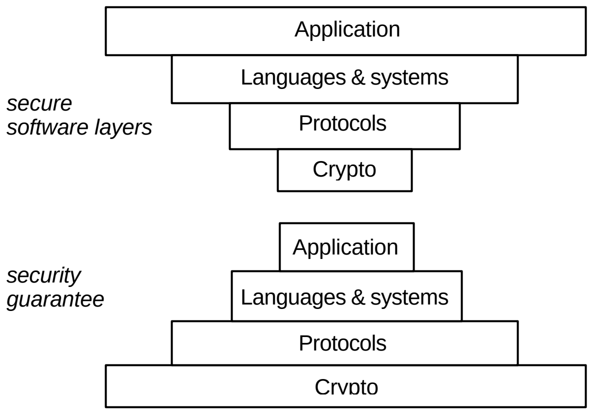

**Fig. 1 – Security problem space vs. security solution space** [Win98]

In this overall context, the MILOS project aims to bring together researchers in requirements engineering [Lam00a, Lam00b], programming languages [Har99, Roy99] and computer security [Joy97, Joy01] to develop an integrated, vertical set of security engineering techniques for the application and language layers –from application-specific security requirements engineering to secure architecture design to secure programming platform implementation (MILOS stands for "Méthodes d'Ingénierie de LOgigicels Sécurisés").

This paper presents some ongoing work on the security requirements engineering side of this project. We start by briefly reviewing in Section 2 some critical (meta-) requirements on development processes for high assurance systems (including security-critical ones). Then we address the problem of specifying security-related requirements and eliciting them in a goal-oriented requirements engineering framework (Section 3). The next section outlines how our analysis techniques for generating and resolving obstacles to goal achievement can be extended to integrate intentional obstacles set up by attackers who want to break security goals; new security requirements are derived from attack generation and resolution. Section 5 illustrates our ideas by rational reconstruction of known attacks and resolutions of web-based banking services.

## 2. Problems & Challenges with High Assurance Systems

High assurance systems are computer systems where compelling evidence is required that the system delivers its services in a manner that satisfies certain critical properties such as safety, security, fault-tolerance and survivability [Lea95]. Our requirements on an ideal development process for such systems are derived more or less trivially from some well-recognized evidence related to this definition.

Evidence 1: The later defects are found, the more expensive and critical they are likely to be.

***Req 1**: High assurance concerns should already be carefully considered at requirements engineeering time so that all required properties related to safety, security, fault-tolerance and survivability (SSFS) are made precise, inter-related for analysis of positive/negative interactions, and preserved at every transition to downstream or companion products.*

Evidence 2: A posteriori detection/resolution of defects may endlessly generate other defects.

***Req 2**: A constructive approach should be adopted where SSFS properties are by construction guaranteed seamlessly throughout system development.*

Evidence 3: High assurance requires a much stronger level of guarantee.

***Req 3**: The elaboration process should be made formal wherever possible and supported by formal analysis tools so that compelling evidence can be provided.*

***Req 4**: The formalism required by Requirement 3 should be lightweight so as to be usable as early as possible in the requirements engineering process.*

Evidence 4: The general objectives of safety, security, fault tolerance and survivability must be refined in a complete and consistent set of application-specific requirements entailing them.

***Requirements 5-8**: The requirements elaboration process should be:*

- *goal-oriented so that SSFS objectives can be guaranteed to be met by lower-level, operational requirements,*
- *incremental so that (a) partial models can be analyzed early and stepwise with respect to the higher-level SSFS objectives, and (b) higher-assurance alternatives can be considered,*
- *relying on rich, multiple models so that all facets of a complex system can be captured and analyzed against SSFS objectives,*
- *open to smooth, roundtrip transitions from informal to semi-formal to formal, and from requirements to architecture so that risks of SSFS leaks during such transitions are minimized.*

Evidence 5: Positive/negative interactions among functional requirements and SSFS-related requirements are more easily analyzed when the requirements specification keeps them separate [Rob03].

***Req 9**: The requirements specification should be structured in a way that keeps functional, safety and security requirements separate from each other.*

## 3. Some Bits of Goal-Oriented Requirements Engineering

We briefly recall some of the concepts and techniques involved in a requirements engineering method we have developed to address those requirements [Lam00a, Lam01] before discussing how such concepts and techniques can be extended to engineer security requirements in a more specific way.

A *goal* is a prescriptive statement of intent about some system (existing or to-be) whose satisfaction in general requires the cooperation of some of the agents forming that system. *Agents* are active components such as humans, devices, legacy software or software-to-be components that play some *role* towards goal satisfaction. Some agents thus define the software whereas the others define its environment. Goals may refer to services to be provided (functional goals) or to quality of service (non-functional goals). Unlike goals, *domain properties* are descriptive statements about the environment –e.g., physical laws, organizational norms, etc.

Goals are organized into AND/OR *refinement-abstraction structures* where higher-level goals are in general strategic, coarse-grained and involve multiple agents whereas lower-level goals are in general technical, fine-grained and involve less agents [Dar93, Dar96]. In such structures, *AND-refinement* links relate a goal to a set of subgoals (called *refinement*) possibly conjoined with domain properties; this means that satisfying all subgoals in the refinement is a sufficient condition in the domain for satisfying the goal. *OR-refinement* links relate a goal to an alternative set of refinements; this means that satisfying one of the refinements is a sufficient condition in the domain for satisfying the goal.

Goal refinement ends when every subgoal is *realizable* by some individual agent assigned to it, that is, expressible in terms of conditions that are monitorable and controllable by the agent [Let02a]. A *requirement* is a terminal goal under responsibility of an agent in the software-to-be; an *expectation* is a terminal goal under responsibility of an agent in the environment (unlike requirements, expectations cannot be enforced by the software-to-be).

Goals prescribe intended behaviors; they can be formalized in a real-time temporal logic [Dar93]. Semi-formal keywords such as Achieve, Avoid, Maintain are provided to name goals according to the temporal behavior pattern they prescribe, without going into temporal logic details. *Softgoals* prescribe preferred behaviors; they can be used for selecting preferred alternatives in an AND/OR goal refinement graph [Chu00].

Goals are *operationalized* into specifications of operations to achieve them [Dar93, Let02b]; in the specifcation of an operation, a distinction is made between *domain* pre- and postconditions that capture what any application of the operation means in the application domain, and *required* pre-, trigger, and postconditions that capture requirements on the operations that are necessary for achieving the underlying goals.

Goals *concern* objects (entities, associations, events, agents) which can be incrementally derived from their formulation to produce a structural model of the system (represented by UML class diagrams).

Agents are related together via their interface made of object attributes they *monitor* and *control*, respectively [Par95].

Obstacles were first introduced in [Pot95] as a way to guide further elicitation through identification of scenarios that could lead to goal violation. In declarative terms, an *obstacle* to some goal is a condition whose satisfaction may prevent the goal from being achieved. More precisely, an obstacle *O* is said to obstruct a goal *G* in a domain characterized by a set of domain properties *Dom* iff

$$\{O, Dom\} \models \neg G \qquad \textit{obstruction}$$

$$Dom \not\models \neg O \qquad \textit{domain consistency}$$

Obstacle analysis consists in taking a pessimistic view at the goals, requirements and expectations elicited in a first stage. The principle is to identify as many ways of breaking them as possible in order to resolve each such situation when critical and likely so as to produce more complete requirements for more robust systems. Formal techniques for generation and AND/OR refinement of obstacles are available [Lam00b]. The basic technique amounts to a precondition calculus that regresses goal negations $\neg G$ backwards through known domain properties *Dom*. Formal obstruction patterns may be used as a cheaper alternative to shorcut formal derivations. Both techniques allow domain properties involved in obstructions to be incrementally elicited as well.

Obstacles need to be resolved once they have been generated. Resolution tactics are available for generating resolutions, notably, *goal substitution*, *agent substitution*, *goal weakening*, *goal restoration*, *obstacle prevention* and *obstacle mitigation* [Lam00b]. Which resolution to select depends on the obstacle criticality, its likelihood of occurrence and on high-priority softgoals that may impact on the selection. The selected resolution may then be deployed at specification time, resulting in specification transformation [Lam00b], or at runtime through specification monitoring that detects or anticipates obstacle occurrences [Fea98].

Obstacle analysis is a recursive process; it may produce new goals for which new obstacles may need to be generated and resolved.

## 4. Preliminary Elicitation of Security Goals

The preliminary elicitation of security-related goals is driven by generic specification patterns associated with each specialization of this goal class, namely, *confidentiality*, *integrity*, *availability* and *privacy* goals. The patterns are expressed in terms of abstractions from the language meta-model and security-specific language constructs. For each subclass, the instantiation of the corresponding specification pattern to "sensitive" objects found in the object model yields corresponding candidates for application-specific security goals (to be refined if necessary).

For example, the specification pattern for Confidentiality goals refers to meta-model elements such as Agent and Object (the latter having a generic attribute Info) to capture the definition of confidentiality [Kem03]:

**Goal** *Avoid* [SensitiveInfoKnownByUnauthorizedAgent]
**FormalSpec** $\forall$ ag: Agent, ob: Object
$\neg$ Authorized (ag, ob.Info) $\Rightarrow$ $\neg$ $KnowsV_{ag}$ (ob.Info)

In this pattern, the Authorized predicate is generic and has to be instantiated through an application-specific domain definition; e.g., for web-based banking services we would certainly consider the instantiation Object / Account while browsing the object model and might introduce the following instantiating definition:

$\forall$ ag: Agent, acc: Account
Authorized (ag, acc) $\equiv$
Owns (ag, acc) $\vee$ Proxy (ag, acc) $\vee$ Manages (ag, acc)

In the pattern above, the epistemic operator *KnowsV$_{ag}$* is a security-specific construct defined by

$\text{KnowsV}_{ag}(v) \equiv \exists x\text{: KnowsV}_{ag}(x = v)$ (“knows value”)

$\text{Knows}_{ag}(P) \equiv \text{Belief}_{ag}(P) \wedge P$ (“knows property”)

where the operational semantics of *Belief$_{ag}$(P)* is “*P is among the properties stored in the local memory of agent ag*”. An agent thus knows a property if that property is found in its local memory and it is indeed the case that the property holds.

For web-based banking services, the “sensitive” information about accounts would be the pair of objects (Acc#, PIN), both partOf the aggregation Account, and linked in the object model through the *Match* association; the instantiation then yields the following confidentiality goal as candidate for inclusion in the goal graph:

**Goal** *Avoid* [PaymentMediumKnownBy3rdParty]
**FormalSpec** $\forall$ p: Person, acc: Account
$\neg$ (Owns (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manages (p, acc))
$\Rightarrow \neg$ [ $\text{KnowsV}_p$ (acc.Acc#) $\wedge$ $\text{KnowsV}_p$ (acc.PIN) ])

The same principle can be applied to elicit instantiations from the generic patterns for Privacy, Integrity and Availability goals, namely:

*Maintain* [PrivateInfoKnownOnlyIfPermittedByOwner]
*Maintain* [ObjectInfoChangeOnlyIfCorrectAndByAuthorizedAgent]
*Achieve* [ObjectInfoUsableWhenNeededByAuthorizedAgent]

In the context of security it may be worth recalling that “sensitive” objects found in the object model can be either passive (entities, associations, events) or active (agents).

## 5. Obstacle Analysis Revisited: Attack Generation and Resolution

As noted in [Lam00b], obstacle refinement trees for security goals correspond to the popular *hazard trees* (sometimes called threat trees or attack trees) used frequently for modeling or documenting known hazards or potential attacks [Amo94, Lev95, Sch99, Moo01, Hel02]. There are two important differences, however.

- Obstacle trees are *goal-anchored*; the analyst thus knows exactly where to start the analysis from -- it is often much easier to concentrate first on what is desired rather than on what is not desired.
- Obstacle refinements are formalized in our real-time temporal logic and can therefore be generated systematically from goal assertions and domain properties using regression techniques or formal refinement patterns [Lam00b].

While obstacle refinement trees may appear sufficient for modeling and resolving accidental, *non-intentional* obstacles to security goals they appear too limited for modeling and resolving malicious, *intentional* obstacles. In the latter case the active attacker should be modelled as well together with his own goals, capabilities, the software vulnerabilities he can monitor or control, etc.

Let us call *anti-models* such richer models and *anti-goals* the intentional obstacles to security goals (we want of course to distinguish them clearly from the goals the system under consideration should satisfy). Anti-models should lead to the generation of more subtle attacks, the discarding of non-realizable or unlikely ones, and the derivation of more effective, “customized” resolutions.

Anti-models can be built systematically as follows.

1. Root anti-goals are obtained just by negation of Confidentiality, Privacy, Integrity and Availability goals instantiated to sensitive objects from the object model as suggested in the previous section.
2. For each anti-goal, potential attacker agents are elicited from questions such as “WHO can benefit from this anti-goal?” (Known attacker taxonomies may help answering such questions.)
3. For each anti-goal and corresponding attacker class(es) identified, the attacker’s higher-level anti-goals are elicited from questions such as “WHY would instances of this attacker class want to achieve this anti-goal?”
4. From the resulting anti-goal graph fragments a *dual* AND/OR refinement/abstraction process can start with the aim of reaching terminal anti-goals that are realizable by the attacker agents identified or by attackee software agents. The former are *anti-requirements* assigned to the attacker whereas the latter are *vulnerabilities* assigned to the attackee. (This rightly fits the Common Criteria definition of *vulnerability*, namely, “a condition of an agent that, in conjunction with a threat, can lead to security requirement violation” [CC99]). The AND/OR refinement/abstraction process can be guided by the following techniques:
   - asking “HOW?” and “WHY?” questions about the anti-goals already available,
   - regressing anti-goal assertions further through domain properties available [Lam00b],
   - applying formal refinement patterns – notably, the milestone and decomposition-by-case pattern [Dar96], the resolve lack of monitorability and

resolve lack of controllability patterns [Let02a] and the obstruction patterns found in [Lam00b].

5. During anti-goal refinement/abstraction the corresponding object and agent anti-models are derived from anti-goal formulations; the boundary and monitoring/control interfaces between the anti-machine (under the attacker's control) and the anti-environment (which includes the software attackee) are thereby derived.
6. Finally, the anti-requirements are AND/OR operationalized in terms of the capabilities of the corresponding attacker agent – which may include blind or intelligent searching, eavesdropping, deciphering, spoofing, etc.)

The above steps result in a multi-component anti-model relating the attackers, their anti-goals, referenced objects and anti-operations to achieve their anti-goals, to the attackees, their goals, objects, operations and vulnerabilities. If goals are formalized in our real-time temporal logic, the logical models of anti-goals are sets of attack scenarios. Bounded SAT solvers can then be used to generate such scenarios automatically.

Once rich attack models have been derived systematically in this way, the last (but not least) step is to exploit them to *explore alternative resolutions* for each derived anti-requirement. A preferred resolution is then selected based on the criticality and likelihood of the corresponding anti-requirement, to produce new security requirements.

A new cycle of anti-goal analysis may then need to be applied to these new requirements. Although we believe that our obstacle resolution operators may be used to generate resolutions, we have not been very far yet in this direction to support that belief. From limited experience on a few small case studies, our feeling is that the *obstacle prevention* operator, resulting in new security goals of form

Avoid [AntiGoal],

frequently yields the most appropriate candidate goals for further refinement.

Other, specific operators can be considered as well such as, e.g.,

- "make vulnerability condition unmonitorable by attacker",
- "make anti-requirement condition uncontrollable by attacker".

For anti-goals that are not too critical, resolutions may be deferred from specification time to run time using intrusion detection technology. We did some experiments recently on this by generating event expressions as input to the ASAX intrusion detection system from anti-goal assertions in temporal logic [Bro02].

## 6. Example: Web-Based Banking Services

We illustrate the critical steps (1) and (4) of the above procedure for web-based banking services and then illustrate what the resolutions might be.

As a result of step (1) we obtain the following anti-goal by negation of the confidentiality goal *Avoid* [PaymentMediumKnownBy3rdParty] obtained by instantiation in Section 3:

**AntiGoal** *Achieve* [PaymentMediumKnownBy3rdParty]
**FormalSpec** $\Diamond \exists$ p: Person, acc: Account
$\neg$ (Owns (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manages (p, acc) )
$\wedge$ $\text{KnowsV}_p$ (acc.Acc#) $\wedge$ $\text{KnowsV}_p$ (acc.PIN)

By asking *"what are sufficient conditions for someone unauthorized to know the number and PIN of an account simultaneously?*" and using the symmetry and [1:1, 1:N] multiplicity of the *Match* association between account numbers and PINs in the object model we elicit domain properties such as:

$\forall$ p: Person, acc: Account
$\neg$ (Owns (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manages (p, acc) )
$\wedge$ $\text{KnowsV}_p$ (acc.Acc#)
$\wedge$ ($\exists$ x: PIN) (Finds (p, x) $\wedge$ Match (x, acc.Acc#)
$\Rightarrow$ $\text{KnowsV}_p$ (acc.Acc#) $\wedge$ $\text{KnowsV}_p$ (acc.PIN)
$\neg$ (Owns (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manages (p, acc) )
$\wedge$ $\text{KnowsV}_p$ (acc.PIN)
$\wedge$ ($\exists$ y: Acc#) (Finds (p, y) $\wedge$ Match (acc.PIN, y)
$\Rightarrow$ $\text{KnowsV}_p$ (acc.Acc#) $\wedge$ $\text{KnowsV}_p$ (acc.PIN)

Regressing the above anti-goal backwards through these domain properties (see [Lam00b]) we obtain an OR-refinement of that anti-goal into two alternative anti-subgoals:

**AntiGoal** *Achieve* [PaymentMediumKnownBy3rdParty FromPinSearching]
**FormalSpec** $\Diamond \exists$ p: Person, acc: Account
$\neg$ (Owns (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manages (p, acc) )
$\wedge$ $\text{KnowsV}_p$ (acc.Acc#)
$\wedge$ ($\exists$ x: PIN) [ Finds (p, x) $\wedge$ Match (x, acc.Acc#) ]

**AntiGoal** *Achieve* [PaymentMediumKnownBy3rdParty FromAccountNumberSearching]
**FormalSpec** $\Diamond \exists$ p: Person, acc: Account
$\neg$ (Owns (p, acc) $\vee$ Proxy (p, acc) $\vee$ Manages (p, acc) )
$\wedge$ $\text{KnowsV}_p$ (acc.PIN)
$\wedge$ ($\exists$ y: Acc#) [ Finds (p, y) $\wedge$ Match (acc.PIN, y) ]

The standard way of resolving the first, well-known anti-goal is to limit the number of PIN entries (e.g., to three attempts). The second anti-goal is more subtle and corresponds to a sophisticated, real attack reported recently [San00]. Resolution through the *obstacle* *prevention* operator would produce the new requirement that the possibility of exhaustive search for account numbers matching some fixed PIN number should be avoided.

No vulnerability anti-goal assigned to the attackee appeared here as conjunct needed to achieve the higher-level anti-goal. In a similar derivation from a confidentiality requirement about orders and personal information in a web-based CD sales system, we derived the vulnerability found in reported attacks on a well-known CD sales company, namely, that order numbers appear on the page URL shown from the order tracking service.

## 7. Related work

A few proposals have been made recently for extending existing modeling notations to capture attacker features at requirements engineering time -- notably, through misuse cases that complement UML use cases [Sin00, Sin01, Ale03] or abuse frames that complement problem frames [Lin03] to define the scope and boundary of anti-requirements on the attacker.

Other extensions of existing frameworks to security requirements analysis include the work by the Toronto group; they propose a methodology based on the i* framework [Yu93] to identify attackers, analyze and propagate vulnerabilities through agent dependency links and suggest counter-measures [Liu03].

The principle of building a catalogue of known threat tree patterns for documentation and reuse through instantiation is nicely illustrated in [Moo01].

Sheyner et al use model checking technology to generate and analyze attack graphs. Given a state machine model *N* of the network under attack, a model *A* of the attacker's capabilities and a desired security property *S*, their tools produce all possible counterexample scenarios found when trying to verify

$$N, A \models S$$

Our work may be seen as an "upstream" complement to their approach; it can be applied sooner in the process to point out earlier security problems at the requirements level by generating attacks from declarative goal/anti-goal models (as opposed to operational state machine models that need to be appropriately built up to reflect such declarative specifications adequately).

## 8. Conclusion

We presented a requirements engineering approach for modeling and specifying security goals and anti-goals, and for deriving attack trees systematically through anti-goal refinement until leaf nodes are reached that are either attackee vulnerabilities observable by the attacker or anti-requirements implementable by this attacker. Anti-goals were seen to be intentional obstacles that obstruct security goals. New security requirements are then derived by resolution of the generated attack trees.

This approach extends our KAOS framework for goal-oriented requirements elaboration by (a) extending the specification language with epistemic constructs for reasoning about the attacker's knowledge, (b) providing a pattern-based approach for the preliminary formal elicitation of security requirements from generic specifications over security-related meta-variables, (c) introducing dual modeling concepts for richer modeling of the attacker's universe, and (d) adapting our goal-oriented method to the elaboration of dual anti-models from which more robust security requirements can be derived. In such dual models, goals, requirements, expectations about the environment and software operations now become intentional obstacles to security requirements, implementable anti-requirements, software vulnerabilities and attack primitives, respectively.

Our approach to attack tree generation is goal-directed; it requires some preliminary elicitation of application-specific security goals, e.g., through instantiation of generic security requirement specifications to application-specific "sensitive" objects. This appears to us the most natural way of coping with security issues at the application level – which is the primary concern of our work.

There are many research issues raised at this point, e.g.,

- Can we apply the same sort of attack tree derivation from *conflict* conditions among multiple goals [Lam98] that can be exploted by attackers?
- What would a rich, comprehensive meta-model for security requirements and anti-requirements look like?
- How do we model trusted agents and incorporate trust models in our framework?
- Can we build rich, reusable libraries of attack patterns [Fon02] and resolution patterns?
- How do we propagate consequences of security leaks in our models?
- How do we incorporate probabilistic reasoning frameworks to reason about partial satisfaction of goals/anti-goals and determine the likelihood of anti-goal occurrences?

These are issues we plan to work on.

**Acknowledgement.** The work reported herein was partially supported by the Belgian "Fond National de la Recherche Scientifique" (FNRS) and the Regional Government of Wallonia (MILOS project, RW Conv. 114856).

## References

[Ale03] I. Alexander, "Misuse Cases: Use Cases with Hostile Intent", IEEE Software, Jan/Feb 2003, 58-66.

[Amo94] E.J. Amoroso, *Fundamentals of Computer Security*. Prentice Hall, 1994.

[Bro02] S. Brohez and Y. Grégoire, *Obstacle Monitoring: an Implementation based on the ASAX Intrusion Detection System*. M.S. Thesis, University of Namur, July 2002.

[CC99] Common Criteria for Information Technology Security Evaluation, Version 2.1, Aug. 1999, http:www.commoncriteria.org/

[CERT] http://www.cert.org/stats/cert_stats.html

[Chu00] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, *Non-functional requirements in software engineering*. Kluwer Academic, Boston, 2000.

[Dar93] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[Dar96] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", *Proc. FSE'4 - Fourth ACM SIGSOFT Symp. on the Foundations of Software Engineering,* San Francisco, October 1996, 179-190.

[Fea98] M. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard, "Reconciling System Requirements and Runtime Behaviour", Proc. IWSSD'98 - 9th International Workshop on Software Specification and Design, Isobe, IEEE CS Press, April 1998.

[Fon02] P.J. Fontaine, Goal-Oriented Elaboration of Security Requirements. M.S. Thesis, Dept. Computing Science, University of Louvain, June 2001.

[Har99] S. Haridi, P. Van Roy, P. Brand, M. Mehl, R. Scheidhauer, and G. Smolka, "Efficient logic variables for distributed computing", *ACM Transactions on Programming Languages and Systems*, 21(3), May 1999.

[Hel02] G. Helmer, J. Wong, M. Slagell, V. Honavar , L. Miller and R. Lutz, "A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System", *Requirements Engineering Journal* Vol. 7 No. 4, 2002, 177-220.

[Joy97] M. Joye and J.J. Quisquater, "On the importance of securing your bins: The garbage-man-in-the-middle attack", in T. Matsumoto, ed., *4th ACM Conference on Computer and Communications Security,* ACM Press, 1997, pp. 135-141.

[Joy01] M. Joye, J.J. Quisquater and Y. Moti, "On the power of misbehaving adversaries and security analysis of EPOC", in *Progress in Cryptology - CT-RSA 2001*, Lectures Notes in Computer Science, Vol. 2020, April 2001.

[Kem03] R.A. Kemmerer, "Cybersecurity", Invited Mini-Tutorial, *Proc. ICSE'03: 25th Intl. Conf. on Software Engineering*, Portland, IEEE Computer Society Press, May 2003, 705-715.

[Lam98] A. van Lamsweerde, R. Darimont, E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering", *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development,* November 1998.

[Lam00a] A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective", *Keynote paper, Proc. ICSE'2000 - 22nd International Conference on Software Engineering,* ACM Press, 2000.

[Lam00b] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE Transactions on Software Engineering,* Special Issue on Exception Handling, October 2000.

[Lam01] A. van Lamsweerde , "Goal-Oriented Requirements Engineering: A Guided Tour", *Invited Minitutorial, Proc. RE'01 - 5th Intl. Symp. Requirements Engineering,* Toronto, August 2001, pp. 249-263.

[Lea95] J. McLean and C. Heitmeyer, "High Assurance Computer Systems: A Research Agenda", America in the Age of Information, National Science and Technology Council Committee on Information and Communications Forum, Bethesda, 1995.

[Let02a] E. Letier and A. van Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", *Proc. ICSE'02: 24th Intl. Conf. on Software Engineering,* Orlando, IEEE Computer Society Press, May 2002.

[Let02b] E. Letier and A. van Lamsweerde, "Deriving Operational Software Specifications from System Goals", *Proc. FSE'10:* 10th *ACM SIGSOFT Symp. on the Foundations of Software Engineering,* Charleston, November 2002.

[Lev95] N. Leveson, *Safeware - System Safety and Computers*. Addison-Wesley, 1995.

[Lin03] L. Lin, B. Nuseibeh, D. Ince, M. Jackson and J. Moffett, "Introducing Abuse Frames for Analyzing Security Requirements", Open University, 2003.

[Liu03] L. Liu, E. Yu and J. Mylopoulos, "Security and Privacy Requirements Analysis with a Social Settinfg", *Proc. RE'03 - International Conference on Requirements Engineering, Monterey*, California, September 2003.

[Moo01] AP. Moore, R.J. Ellison and R.C. Linger, "Attack Modeling for Information Security and Survivability", Technical Note CMU/SEI-2001-TN-001, March 2001.

[Par95]D.L. Parnas and J. Madey, "Functional Documents for Computer Systems", *Science of Computer Programming,* Vol. 25, 1995, pp. 41-61.

[Pot95] C. Potts, "Using Schematic Scenarios to Understand User Needs", *Proc. DIS'95 - ACM Symposium on Designing interactive Systems: Processes, Practices and Techniques*, University of Michigan, August 1995.

[Rob03] W. N. Robinson, “Requirements Interaction Management”, *ACM Computing Surveys*, June 2003.

[Roy99] P. Van Roy, P. Brand, S. Haridi, and R. Collet, “A lightweight reliable object migration protocol”, *Lecture Notes in Computer Science*, vol. 1686, Springer-Verlag, October 1999.

[San00] A. dos Santos, G. Vigna, and R. Kemmerer, “Security Testing of the Online Banking Service of a Large International Bank”, *Proceedings of the First Workshop on Security and Privacy in E-Commerce*, November 2000.

[Sch99] B. Schneier, “Attack Trees: Modeling Security Threats”, Dr. Dobb’s Journal, December 1999.

[Sch00] B. Schneier, *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2000.

[She02] O. Sheyner, J. Haines, S. Jha, R. Lippmann and J. Winf, “Automated Generation and Analysis of Attack Graphs”, *Proc. IEEE Symp. on Security and Privacy*, Oakland (CA), May 2002.

[Sin00] G. Sindre and A.L. Opdahl, “Eliciting Security Requirements by Misuse Cases, *Proc. TOOLS Pacific’2000*, Conf. on Techniques of Object-Oriented Languages and Systems, 2000, 120-131.

[Sin01] G. Sindre and A.L. Opdahl, “Templates for Misuse Case Description”, *Proc. REFSQ’01 – Intl. Workshop on Requirements Engineering: Foundations for Software Quality*, 2001.

[Vie01] J. Viega and G. McGraw, *Building Secure Software: How to Avoid Security Problems the Right Way.* Pearson Education, 2001.

[Yu93] E.S.K. Yu, "Modelling Organizations for Information Systems Requirements Engineering", *Proc. RE'93 - 1st Intl Symp. on Requirements Engineering*, IEEE, 1993, 34-41.

[Win98] J. Wing, "A Symbiotic Relationship Between Formal Methods and Security", Proc. NSF Workshop on *Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solution*. December 1998.