On the Fly Design and Co-simulation of Responses Against Simultaneous Attacks

Léa Samarji[1,2(✉)], Nora Cuppens-Boulahia[1], Frédéric Cuppens[1], Serge Papillon[2], Waël Kanoun[2], and Samuel Dubus[2]

[1] Télécom Bretagne, rue de la Chataigneraie, 35510 Cesson-Sévigné, France
{layal.elsamarji,nora.cuppens,frederic.cuppens}@telecom-bretagne.eu
[2] Alcatel-Lucent Bell Labs, Villarceaux, route de Villejust, 91620 Nozay, France
{serge.papillon,wael.kanoun,samuel.dubus}@alcatel-lucent.com

Abstract. The growth of critical information systems in size and complexity has driven the research community to propose automated response systems. These systems must cope with the steady progress of the attacks' sophistication, coordination and effectiveness. Unfortunately, existing response systems still handle attacks independently, suffering thereby from (i) efficiency issues against coordinated attacks (e.g. DDoS), (ii) conflicts between parallel responses, and (iii) unexpected side effects of responses on the system. We, thus, propose in this paper a new response model against simultaneous threats. Our response is dynamically designed based on a new definition of capability-aware logic anti-correlation, and modeled using the Situation Calculus (SC) language. Even though a response can prevent or reduce an attack scenario, it may also have side effects on the system and unintentionally ease one of the attackers to progress on its scenario. We address this issue by proposing a response co-simulator based on SC planning capabilities. This co-simulator considers each response candidate apart and reasons, from the current system's and attackers' state, to assess the achieved risk mitigation on the protected system. Experimentations were led to highlight the benefits of our solution.

Keywords: Response system · Simultaneous attacks · Situation calculus

1 Introduction

Modern attack tools are rapidly evolving to become more powerful and sophisticated. Networks and information systems are frequently targeted by simultaneous attacks, which causes deterioration in system's performance and induce great damage to physical assets. Simultaneous attacks are those performed by different attack entities. Each of them may be a single individual attacker or composed of a Group of Coordinated Attackers (GCA), with a specific attack objective in the system. When the attack entity is a GCA, the system risks to suffer from coordinated attacks [20]. Unfortunately, existing response systems

G. Pernul et al. (Eds.): ESORICS 2015, Part II, LNCS 9327, pp. 642–661, 2015.
DOI: 10.1007/978-3-319-24177-7_32

proposals [7–9,17,19] still handle attacks as being independent actions, and each attack scenario is treated as if it is the only intrusion scenario in the system. Moreover, the majority of automated intrusion response systems rely on a predefined mapping of response actions to attacks. While this approach allows a system administrator to deal with intrusions faster, it lacks flexibility as "things do not always turn out the way we planned". Besides, when responding to simultaneous attacks by activating parallel response measures, unexpected conflicts between responses may occur, in addition to potential side effects on the system.

In this paper we propose a response system that overcomes these problems by dynamically designing and co-simulating response candidates for simultaneous attack threats. We first introduce a new response scheme. Our response is described as a sequence of non conflicting parallel actions, allowing thereby an execution in parallel or in sequence of different actions handling all the risky threats. Our response is dynamically designed based on a new definition of a capability-aware logic anticorrelation approach [2], and modeled through an efficient logic language, the Situation Calculus (SC) [11,14].

When a system is simultaneously threatened by different attack entities, multiple response candidates may be dynamically designed. In order to choose the most effective response, we also present in this paper a co-simulator based on the SC planning capabilities. This enables to have a real-time estimation of the total risk mitigation induced by each response candidate, and thus, the most optimal one can be chosen and deployed in the system.

The paper is organized as follows: Sect. 2 presents a Simultaneous Attacks Graph (SAG) which forecasts potential simultaneous attack scenarios. SAG is an input for our framework, since it allows us to identify risky threats for which we have to design a response. Section 3 introduces our new response scheme, and explains how responses can be dynamically designed based on a new definition of anticorrelation. Section 4 introduces the Situation Calculus language, and shows how to model our dynamic response with SC. Section 5 introduces our response co-simulator based on SC planning capabilities. In Sect. 6, we experiment our framework on an IP network threatened by simultaneous attacks. Then, Sect. 7 discusses related works, before we conclude on our proposals in Sect. 8.

2 Simultaneous Attacks Graphs

[15] differs from other works on attack modeling (e.g. [1,4,18], etc.), by proposing a formal description of actions that correspond to all types of attacks (individual, coordinated and simultaneous ones). An algorithm was also proposed to generate Attack Graphs (SAG) corresponding to Simultaneous attacks scenarios. Hence, given a system's state and a set of suspicious attackers, the algorithm is able to generate multiple SAGs each corresponding, to a combination of scenarios predicted for those attackers. Figures 1 and 2 are two examples of generated graphs for a telephony operator threatened by fourteen suspicious attackers at time $t0$. In each graph, attackers may be assembled differently into groups, and may

have a different end goal. Consequently, each graph represents only one combination of potential scenarios that can be simultaneously performed in the future by suspicious attackers. In the SAG of Fig. 1, the algorithm assembles attackers into 3 groups of coordinated attackers $\{a1, a2, a3, a4, a5, a6, a7, a8, a9, a10\}$, $\{a11, a12\}$, and $\{a13, a14\}$. For the first group, a scenario targeting the operator's reputation on its principal services (e.g. QoS) is predicted by performing a Distributed Denial of Service (DDoS) on one of its SIP servers. For the second group, a scenario that aims at inducing losses for an operator's client is predicted by hijacking its account and performing a toll fraud. For the third group, a scenario that aims at inducing losses for an operator's political client is predicted by recording its conversation and using it for black mailing. In another example, in Fig. 2, attackers $a11$ and $a12$ are not coordinated, and each one has it own end goal. Notice that an attack entity (e.g. $a11$) can block another one (e.g. $a12$) from progressing. This can occur due to a simultaneous access to unshareable resources in the system. Attackers in this case are called concurrent.

In [16], a new framework was proposed to properly assess the risk of Individual, Coordinated, and Concurrent Attack Scenarios. The method is based on a coordination aware-*Game Theoric* approach to derive an *Attack Likelihood* equation. An algorithm was also proposed to assess the *risk* of each attack scenario in a given SAG, considering the concurrency between attackers. Consequently, the SAG containing scenarios that are the most risky (i.e. scenarios with high likelihood and high impact) in the system is chosen. The most risky SAG is an input for our framework. Our framework will dynamically design and co-simulate a response efficient against attackers leading the risky scenarios within this SAG.

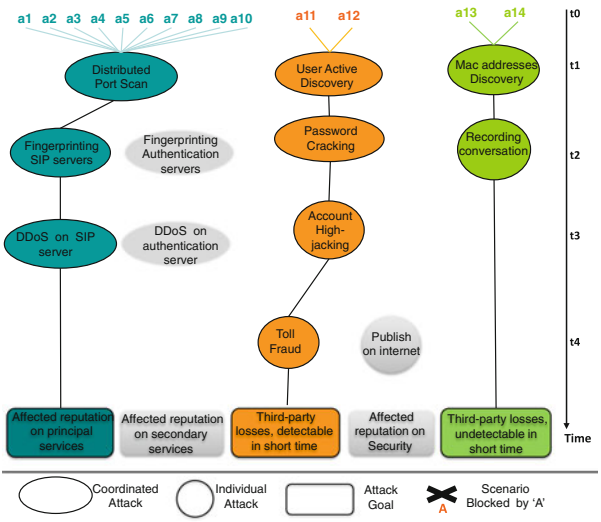

Fig. 1. Example 1 of a VoIP SAG.

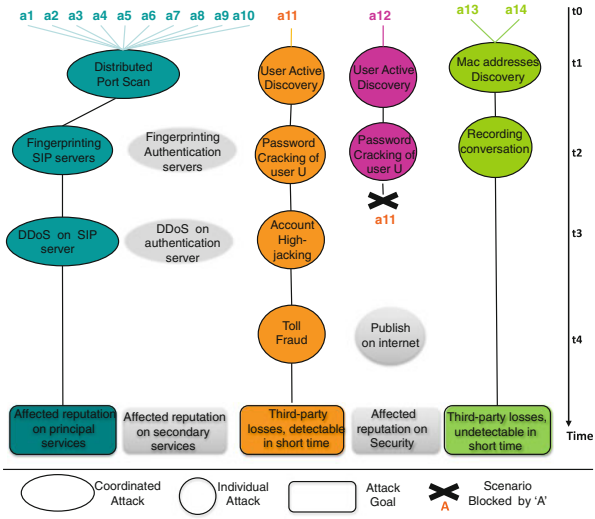

Fig. 2. Example 2 of a VoIP SAG.

3 New Scheme for a Complex Response

Let $[[a_1^1, a_2^1, ..., a_M^1]; [a_1^2, a_2^2, ..., a_M^2]; ...; [a_1^N, a_2^N, ..., a_M^N]]$ be the sequence of simultaneous attacks forecasted in the input SAG, where M different attack entities are presented in the system, with a_j^i being the action that is going to be executed in the i^{th} place by the attack entity j. Symbol $'$,$'$ represents parallelism, and symbol $'$;$'$ represents sequencing. Note that a_j^i can also be no operation if no action is predicted for the entity. Consider that attack entities $1, 2, ..., K$ are considered as risky and attack entities $K+1, ..., M$ are considered as not risky. Note that, in a given SAG, risky attack entities are those for which risky scenarios are forecasted. Hence, the following is the sequence of Risky Simultaneous Attacks ($RiskySAS$): $RiskySAS = [[a_1^1, ..., a_K^1]; [a_1^2, ..., a_K^2]; ...; [a_1^N, ..., a_K^N]]$.

R is considered a response against $RiskySAS$, if R is able, while maintaining the system in an operational state, to either prevent or delay entities $1, 2, ..., K$ from reaching their attack objectives. In order to respond to simultaneous threats, the response should not be limited to a single elementary action, we thus consider a response as a complex action, and we define it as a set of partially ordered elementary actions. In other words, a response may consist of non conflicting system's actions activated in parallel and of system's actions activated in sequence. Contrarily to an attack action, a system's action is an action triggered/executed by the system. $openSession(Src\_ip, Src\_port, Dest\_ip, Dest\_port)$, $restart(Server)$, $install(patchID, machineIP)$, $deploy(StrongAuthentication, Server)$, are examples of system's actions.

We introduce, in the following, a generic action scheme for a response R described as a sequence of length x of parallel system's actions: $R = [[r_1^1, ..., r_{l1}^1]; [r_1^2, ..., r_{l2}^2]; ...; [r_1^x, ..., r_{lx}^x]]$ with r_i^k being one of the lk system's actions

executed in the k^{th} place (i.e. k^{th} time step) , and $\forall k/\ 1 < k < x, \forall i \neq j\ /\ 1 < i, j < lk$, r_i^k and r_j^k are not conflicting (i.e. meaning that parallel actions should be compatible together for a parallel execution).

In order to design such a response on the fly against a *RiskySAS*, a dynamic anticorrelation logic approach should be applied. Unfortunately, existing work on anticorrelation [2] is limited to an anticorrelation definition unaware of the applicability of actions, and is thereby inefficient for dynamic use. Moreover, the existing definition of anticorrelation is limited to responses consisting of a single elementary action. We thus propose in the following sections an applicability-aware anticorrelation definition adapted to our response scheme.

3.1 Applicability-Aware Anticorrelation

Anticorrelation in logic programming was defined in [2] as follows:

Definition 1. *Let r and a be respectively a system's action and an attack. postr is the set of predicates of post-condition of r and prea is the set of predicates of pre-condition of a. Each of the post-condition and the pre-condition is a conjunction of predicates. r and a are anti-correlated if the following is satisfied:* $anticorrelated(r, a) \leftrightarrow \exists Pr, Pa/(Pr \in postr \wedge \neg Pa \in prea) \wedge$ *Pr,Pa are unifiable.*

Consider $passwordCrack(Attacker_1, U, Serv)$ a cracking of user U's password through server $Serv$. A precondition of this attack is to have $Attacker_1$ having network access to $Serv$. Let $discard(Attacker_1, Serv)$ be a system's action consisting in disconnecting $Attacker_1$ from $Serv$'s network. Consequently, we have: $anticorrelated(discard(Attacker_1, Serv), passwordCrack(Attacker_1, U, Serv))$.

Unfortunately, Definition 1 does not consider the applicability of the system's action. Actually, if we reconsider the latest example, the $discard(Attacker_1, Serv)$ action, may not be possible for execution in the current state because the database containing allowed ip addresses for connection to $Serv$ is exclusively opened by another module in the system. Hence, response system should wait until the database is released, to be able to execute its action. Therefore, we propose an applicability-aware anticorrelation definition as follows:

Definition 2. *Let r and a be respectively a system's action and an attack. Let S be the current system state, and poss(r,S) a predicate meaning that it is possible to execute r in S. r and a are anti-correlated in S if the following is satisfied:* $anticorrelated(r, a, S) \leftrightarrow poss(r, S) \wedge anticorrelated(r, a)$.
$poss(r, S) \leftrightarrow \forall P \in prer,$ *P holds in S.*

Definition 2 is limited to responses consisting of a single elementary action. However, a system may sometimes need to coordinate multiple elementary actions in order to react against an attack a, especially when a is a coordinated attack [15]. As an example, consider that 25 users are registered to server $Serv$, that can handle up to 20 users deploying their entire bandwidth. Consider now that 22 users were infected by an external bot, and they are coordinately

flooding *Serv* (i.e. executing a DDoS). The set of compromised users is thus considered as a GCA. In order to respond to the DDoS attack, the system should discard in parallel at least two of the infected users to reduce the receiving flow below the threshold of *Serv*. Thus, it is the resulting effect of the three *discard* actions which is opposite to the precondition of DDoS (which is $\mid GCA \mid > 20$). The following is a logic expression of the combined effect of the three elementary actions: $discarded(User_{21}) \wedge discarded(User_{22}) \rightarrow \mid GCA \mid < 20$.

In this case, it is the set of parallel actions $[discard(User_{21}), discard(User_{22})]$ which is anticorrelated with $DDoS(GCA, Serv)$. Therefore, we propose a definition of applicability-aware anticorrelation between a set of coordinated elementary system's actions and an attack, as follows:

Definition 3. *Let* $rcoordinated = [r_1, r_2, ..., r_C]$ *be a set of parallel system's actions, and a an attack. postrk is the set of predicates of post-conditions of* r_k *and prea is the set of predicates of pre-condition of a. rcoordinated and a are anti-correlated if the following condition is satisfied:*
$anticorrelated([r_1, r_2, .., r_C], a, S) \leftrightarrow$
$poss([r_1, r_2, .., r_C], S) \wedge anticorrelated([r_1, r_2, .., r_C], a)$. *with:*
$anticorrelated([r_1, r_2, .., r_C], a) \leftrightarrow (Pr_1 \wedge Pr_2 \wedge .. \wedge Pr_C \rightarrow Pr)\ / Pr_k \in postr_k \wedge \exists Pa\ /(\neg Pa \in prea \wedge Pr, Pa$ *are unifiable).* *and*
$poss([r_1, r_2, ..., r_C], S) \leftrightarrow \forall i \in [1, C], poss(r_i, S) \wedge \forall j \neq i / 1 < j < C, \neg conflict(r_i, r_j)$.

$\neg conflict(r_i, r_j)$ means that r_i and r_j are not conflicting (i.e. can be executed simultaneously). The semantic definition of *conflict* will be given in Sect. 4.3.

Definition 3 can be extended to the case of a complex system action (i.e. a sequence of parallel system's actions) as follows:

Definition 4. *Let* $R* = [[r_1^1, ..., r_{l1}^1]; [r_1^2, ..., r_{l2}^2]; ...; [r_1^c, ..., r_{lc}^c]]$ *be a complex action, and a an attack action. Let* S_{i+1} *be the state of the system after the execution of* $[r_1^i, ..., r_{li}^i]$ *in state* S_i. *R* and a are anticorrelated in state* S_{c-1} *if the following condition is satisfied:*
$anticorrelated([[r_1^1, ..., r_{l1}^1]; [r_1^2, ..., r_{l2}^2]; ...; [r_1^c, ..., r_{lc}^c]], a, S_{c-1})$
$\leftrightarrow anticorrelated([r_1^c, ..., r_{lc}^c], a) \wedge poss([r_1^1, ..., r_{l1}^1], S_0) \wedge$
$poss([r_1^2, ..., r_{l2}^2], S_1) \wedge .... \wedge poss([r_1^c, ..., r_{lc}^c], S_{c-1})$.

Based on Definition 4, we now propose a definition of applicability-aware anticorrelation between a complex action R, and a *RiskySAS* as follows:

Definition 5. *Let* $R = [[r_1^1, ..., r_{l1}^1]; [r_1^2, ..., r_{l2}^2]; ...; [r_1^x, ..., r_{lx}^x]]$ *be a complex action, and RiskySAS a set of risky simultaneous attack scenarios, such that* $RiskySAS = [[a_1^1, ..., a_K^1]; [a_1^2, ..., a_K^2]; ...; [a_1^N, ..., a_K^N]]$, *with* a_j^i *being an attack performed by attack entity j. R and RiskySAS are anti-correlated if the following condition is satisfied:*
$anticorrelated(R, RiskySAS, S) \leftrightarrow \forall entity(j), \exists R* \in R, \exists a \in [a_j^1; a_j^2; ...; a_j^N] /$
$anticorrelated(R*, a, S)$.

Thus, R is anticorrelated with $RiskySAS$ if for each attack sequence corresponding to an attack entity, we can find a complex action $R*$ within R which is anticorrelated and applicable with at least one of the attacks that the entity will execute throughout its sequence.

3.2 Complex Response

A complex action R is considered a dynamic response against $RiskySAS$ in a state S, if R is applicable in S and anticorrelated with $RiskySAS$, and no nominal constraint is violated at the end of R's execution. Nominal constraints are those related to critical system assets that should not be violated, in order to guarantee a minimum operating state (i.e. service continuity) in the system. Consequently, R should include 'operability' actions whenever nominal constraints risk to be violated by the system's actions composing R. We, thus, define a response R against a $RiskySAS$ as follows:

Definition 6. *Let* $R = [[r^1_1, ..., r^1_{l1}]; [r^2_1, ..., r^2_{l2}]; ...; [r^x_1, ..., r^x_{lx}]]$ *be a complex action, and RiskySAS a set of risky simultaneous attack scenarios. And, let min_constraints be the set of nominal constraints. R is a response against RiskySAS in state S if the following condition is satisfied:*
$response(R, RiskySAS, S) \leftrightarrow$ *anticorrelated(R,RiskySAS,S)*
$\wedge \forall Constraint \in min\_constraints, Constraint(S) = True.$

For example, consider a system threatened simultaneously by two risky threats T1 and T2. T1 aims at over-flooding server S1, and T2 aims at hijacking a legitimate user's account U throughout a machine M infected with a bot. The following sequence of parallel actions corresponds, thus, to a response against T1 and T2.

$$R = [\,[shareLoad(S1, S2), disconnect(M)]; [backupFiles(M, BackupServer)];$$
$$[reformatHarddrive(M)]; [install(SecurityPatch, M)]; [connect(M)]\,].$$

In a first step, sharing load is settled between $S1$ and another server $S2$ in order to prevent $S1$ from being overcharged by T1. In parallel, M is disconnected from the network, and files on M are backed up in order to reformat the machine and install a security patch in further steps. By this, the bot on M is removed and the vulnerability is patched. In this example, $shareLoad(S1, S2)$, and $install(SecurityPatch, M)$ are two system's actions anticorrelated respectively with T1 and T2. *disconnect(M)*, *reformatHarddrive(M)* and *backupFiles(M, BackupServer)* are actions rendering the system capable to execute *install(SecurityPatch,M)*. We call them 'capability enabling' actions. Finally, *connect(M)* is an 'operability' action. Note that without this latter, the response against T2 will not be effective.

In order to design our responses based on logic anticorrelation, system and attacks actions should be modeled using the same logic language. Additionally, the modeling language should follow a pre/post condition approach for actions description. In [15] different modeling languages were compared: LAMBDA [4],

STRIPS [1], JIGSAW [18], and Situation Calculus [11,14], and the latter turns to be the most adapted language to describe all attack types. We, thus, investigate in the next section the adaptability of SC in modeling anticorrelation and responses as defined in the previous section.

4 Modeling Responses with Situation Calculus (SC)

4.1 Basics of the Situation Calculus

Situation Calculus [11,14] is a dialect of first order logic, with second order-logic terms for representing dynamic change. It basically consists of:

- Situations: a situation represents the system's state, and the action's history (i.e. sequence) from an initial empty action sequence $S0$.
- Fluents and Predicates: the world is described in terms of predicates and fluents. Whereas predicates are stateless, fluents are statefull, and thus take situations as arguments. For example, *Server*(*Serv*) is a predicate meaning that *Serv* is a server. While, *network_access*(*M1, S2, s*) is a fluent meaning that machine *M1* has a network access to server *S2* in situation *s*. Additionally, Fluents can be either relational, or functional. Relational fluents return boolean values, e.g. *is_on*(*Serv, s*), while functional fluents return a non boolean value, e.g. *received_flow*(*Serv,s*)=*500*.
- Actions: consist of a function symbol and its arguments. For example, *reboot*(*Server1*) is the action of rebooting *Server1*. In order to reason about the effects of an action, we refer to function *do*(*a,s*). This latter denotes the situation that results from doing action *a* in situation *s*.

SC also provides essential axioms to represent dynamic changes:

- Action precondition axioms: for each action a, there is a predicate $Poss(a, s)$ that states if it is possible for action a to be executed in situation s.
- Successor state axioms: there is one for each fluent F. It characterizes the conditions under which a fluent $F(x, do(a, s))$ changes from s to $do(a, s)$.

4.2 Elementary System Actions

SC answers our need in (1) offering the possibility to dynamically design a response whose requirements and effects depend on the system's state, and (2) modeling system actions following a pre/post condition approach. The following is an SC description of an operational action ***shareLoad***$(S1, S2)$ which consists in forcing a server $S1$ to share the load with another server $S2$ when overcharged.
$\mathbf{Poss}(shareLoad(S1, S2), S) \leftrightarrow overcharged(S1, S) \ \wedge is\_on(S2)$
$\wedge runningService(S1, S) = runningService(S2, S)$.
$overcharged(S1, \mathbf{do}(A, S)) \leftrightarrow A = ddos(GCA, S1) \vee (overcharged(S1, S) \wedge A \neq shareLoad(S1, S2))$.

4.3 Concurrent System Actions

In [12,13], SC was expanded to handle concurrency. A new sort *concurrent* is added. Every *concurrent* variable c is a set of concurrent simple actions a. The binary function $do(c, s)$ returns a situation term that results from the application of concurrent actions c in situation s. $Poss(a, s)$ is thus extended to $Poss(c, s)$.

Additionally, in a simultaneous actions context, some actions can not be performed concurrently. This is due to incompatibility between actions in terms of resources that each action uses. As a solution, Pinto [12] proposed to add a finer level of granularity by appealing to the notion of resource: $xres(a, r)$ means that the action a requires the exclusive use of the resource r, and $sres(a, r)$ means that the action a requires the use of the resource r for its execution, but r can be shared. Finally, $poss(c, s)$ makes use of a conflict predicate $conflict$ as a precondition in order to test compatibility between actions:

$$conflict(c) \leftrightarrow \exists a_1, a_2 \in c, \exists r \mid [(xres(a_1, r) \wedge xres(a_2, r)) \vee (xres(a_1, r) \wedge sres(a_2, r)) \vee (sres(a_1, r) \wedge xres(a_2, r))]$$

$$Poss(c, s) \leftrightarrow [\forall a \in c, Poss(a, s)] \wedge \neg conflict(c)$$

Concurrent SC is thus efficient to avoid conflicts while designing a response.

4.4 Anticorrelation and Response in Situation Calculus

Let r and a be respectively a SC description of a system's action, and an attack action. Anticorrelation between r and a presented in Definition 2 is expressed in SC as follows: $anticorrelated(r, a, S) \leftrightarrow poss(r, S) \wedge \neg poss(a, do(r, S))$.

In SC, doing action r renders action a not possible for execution, this means that r has rendered one of a's precondition's predicates unfulfilled (i.e. false).

Let $C = [r_1, r_2, ..., r_L]$ be a set of parallel system's actions, and a an attack action. Anticorrelation between *rconcurrent* and a, as presented in Definition 3, can be expressed in concurrent SC as follows:
$anticorrelated(C, a, S) \leftrightarrow poss(C, S) \wedge \neg poss(a, do(C, S))$.

Now, let $R* = [C_0; C_1; ...; C_k]$ be a complex action with $C_i = [r_1^i, ..., r_{li}^i]$, and a an attack action. Anticorrelation between $R*$ and a, as presented in Definition 4, can be expressed in concurrent SC as follows:
$anticorrelated(R*, a, S) \leftrightarrow poss(R*, S) \wedge \neg poss(a, do(R*, S))$. with
$poss(R*, S) \leftrightarrow poss(C_0, S_0) \wedge poss(C_1, do(C_0, S_0)) \wedge ... \wedge$
$poss(C_k, do(C_{k-1}, (..(do(C_0, S_0)..))$.

Consequently, concurrent SC is adapted to model anticorrelation of a complex action against a *RiskySAS* as defined in Definition 5. And, a response (see Definition 6) can be modeled in SC as follows:
response(R,RiskySAS,S) $\leftrightarrow$ *anticorrelated(R,RiskySAS,S)*$\wedge$
$\forall Ct \in min\_constraints, Ct(S)$. *//constraints can be modeled in SC as shown in* [5].

At this stage of the paper, we have proposed a mean to dynamically design a response against a set of simultaneous attacks scenarios. Since multiple response possibilities may exist, we introduce in the next section, the SC planning task that we use to propose a dynamic response co-simulator.

5 Planning in Situation Calculus

In [14], the author presented and implemented the world's simplest breadth-first planner ($wspbf$). $wspbf$ is a SC planner for an agent who can perform concurrent or sequential actions. It is supplied with a goal predicate $plannerGoal(s)$ to fulfill. Here is the Golog [10] program of the $wspbf$:

proc $wspbf(n)$
$plans(0, n)$
endProc

proc $plans(m, n)$
$m \leq n?; [actionSequence(m); plannerGoal? \mid plans(m+1,\ n)]$
endProc

proc $actionSequence(n)$
$n = 0? \mid n > 0?; (\pi c)\,[concurrent\_actions(c)?; c]\,; actionSequence(n\text{-}1)$
endProc

The planner generates all sequences of concurrent actions c fulfilling the goal. It terminates with failure if it does not find a sequence, which length is smaller or equal to n, that fulfills the goal.

5.1 Dynamic Response Co-simulator Based on SC Planning

We generalize the $wspbf$ to the case of a multi-agent system, where we have, on one hand, the system which can perform, concurrently or sequentially, a set of actions, and on the other hand, the attack entities present in the SAG, which can perform individual or coordinated attacks. First, we integrate all the attacks that have been specified to generate the SAGs. Second, a network and a security expert are needed to specify and describe in SC, all the elementary actions that the system can perform, considering the resource notion. Third, we integrate all the attack goals that have been specified to generate the SAGs. For instance, the following are two critical assets that may be considered attack goals in a system handling a voice over IP (VoIP) service.

$Attack\_Goal(Entity,S) \rightarrow in\_denial(Entity, VoIPserver, S) \vee is\_off(Entity, VoIPuser, S)$.
//meaning that an attack entity can reach an attack goal in situation S, if in S, it has succeeded a denial of service over a VoIP server or a VoIP user.

Forth, we describe more specifically the attack goal that each attack entity has reached in the considered SAG. For example, if $entity1$ has overflown a VoIP server then: $goalreached(entity1, S) \leftrightarrow in\_denial(entity1, VoIPserver, S)$.

Fifth, we specify in SC, for each attack entity appearing in SAG, if it is risky or not [16]. Besides, we specify for each risky entity, its attack scenario. For instance, $risky(entity1) \wedge riskySAS(entity1, scenario1)$, and $\neg risky(entityM)$.

Finally, we configure the co-simulator goal in a manner to reach a situation where a response is designed based on described system actions, and every risky entity is either (1) completely prevented from reaching her attack goal, or (2) forced to change her path and choose a more complex one before getting to her goal, thereby, reducing her risk. Concerning non risky entities, since they are not the prior concern of the system in the current situation, then no response will be intentionally designed for them. Note that if a response was able to additionally block or reduce the risk of a non risky entity, then this is also considered a solution for our co-simulator. We model our co-simulator's goal as follows:

$plannerGoal(S) \rightarrow$
$\forall\, risky(EntityA), [\, riskBlocked(EntityA, S) \vee riskReduced(EntityA, S)] \wedge$
$\forall \neg risky(EntityB), [\, riskBlocked(EntityB, S) \vee riskReduced(EntityB, S)$
$\vee\, Attack\_Goal(EntityB, S)].$ with:

$riskBlocked(Entity, do(R*, S)) \rightarrow \exists Scenario/$
$riskySAS(Entity, Scenario) \wedge response(R*, Scenario, S) \vee riskBlocked(Entity, S).$
//meaning that: due to the response R, the attack entity was completely prevented from performing her attack scenario. Thus, the risk of this entity is totally blocked.*

$riskReduced(Entity, S) \rightarrow goalReached(Entity, S) \wedge privilegesLoss(Entity, S).$
$privilegesLoss(Entity, do(C, S)) \rightarrow [\exists Predicate, \exists Object/Predicate(Entity,Object,S)$
$\wedge \neg Predicate(Entity,Object,do(C,S))] \vee privilegesLoss(Entity, S).$
//meaning that: due to some system actions C making part of the response, the attack entity has lost one of its privileges. Consequently, the entity will need to do more effort, thus, more time, to progress in its scenario Hence, the risk of this entity will decrease.

Our response co-simulator generates an exhaustive list of all the response possibilities that can be designed against the risky threats, co-simulating, for each response, the potential behavior of the attackers face to this response, and the side effects that this latter can have on the system. Note that, each of the generated responses appears within a response plan. A response plan is a sequence of parallel actions. Each action can be either an attack or a system action. Actions in sequence are ordered in time, thereby, an administrator knows when to execute each system action making part of the response.

6 Experimentation

We implemented our response co-simulator using a prolog interpreter, SWI prolog (http://www.swi-prolog.org/). Then, we considered two different use cases for experimentation. In the first, we highlight the capability of our framework in generating responses handling sequencing and parallelism, and simulating the responses side effects. In the second experimentation, we highlight the efficiency of our framework in managing the conflict between actions.

6.1 Use Case 1

In Use case 1, we consider two simultaneous threats led by two attack entities (A1 and A2), as shown in the SAG of Fig. 3. In the initial system state, A1 has already infected machine M1 and actively scanned user U. In parallel, A2 has already infected machine M2 which belongs with M1 to the same Ethernet network (machines are reachable via $Switch12$). It is predicted for A1 to crack the password of U's account and highjack it in order to do a toll fraud which induces economic losses to U. Besides, a likely scenario for A2 is predicted starting by discovering M1 and then poisoning it with ARP messages, in order to spoof its address later on and make calls or inject packets as if they were sent by M1.

In a first experimentation, we consider that both threats are risky. Thus, our planner derives response plans for both of them. The following sequences are some of the response plans proposed by our planner:

Experimentation 1 - Response plan 1:

t1: [[$passCrack(A1, server, u), discovermacaddress(A2, M2, M1)$];
t2: [**notifychangepassword(u,server)**, **deployAuthentication (Switch12)**];
t3: [$passCrack(A1, server, u)$];
t4: [$highjack(A1, u, server)$];
t5: [$tollFraud(A1, u)$]]

Plan 1, presented in the graph of Fig. 4, designs a response R2 against both threats as parallel system actions. The first notifies U to change his password, and the second deploys an authentication on $Swith12$. Due to changing U's password, A1 is no more able to highjack U's account. Thus A1 should re-execute again a password cracking in order to continue its scenario. R2 is considered a response against A1, because it delays A1 from reaching its attack goal, thereby reducing its risk. Due to deploying authentication on switch12, A2 will not be able to poison M1 with ARP messages. Thus, it will not be able to reach its attack goal.

Experimentation 1 - Response plan 2:

t1: [[**disconnect(M1)**];
t2: [**install(SecurityPatch,M1)**];
t3: [**connect(M1)**];
t4: [$discovermacaddress(A2, M2, M1)$];
t5: [$arppoisoning(A2, M2, M1)$];
t6: [$injectRTPpackets(A2, M2, M1)$]]

Plan 2, presented in the graph of Fig. 5, designs a response R1 against both threats as a sequence of system actions. The response consists in patching the vulnerability in M1 and thereby prohibiting A1 from having a remote access to this machine. By disconnecting M1 in order to patch it, A2 will not be able to discover the mac address of M1. Consequently A2 will have to wait until M1 is reconnected to the network in order to continue its scenario. Hence, R1 blocks completely A1, and reduces the risk of A2.

In a second experimentation, we now consider that A1 is a risky threat, whereas A2 is not risky yet. Hence, we configure the planner's goal to respond against A1. The following is one of the solutions proposed by our planner:

Experimentation 2 - Response plan 1:

t1: [$[passCrack(A1, server, u), discovermacaddress(A2, M2, M1)]$;
t2: [**disconnect(M1)**];
t3: [**install(SecurityPatch,M1)**, $injectRTPpackets(A2, M2, M1)$];
t4: [**connect(M1)**]]

The above sequence, presented in the graph of Fig. 6, designs a response R3 against threat A1. R3 consists in patching the vulnerability of M1, and blocking thereby A1. Note that, R3 is composed of the same actions as for R1. The only difference is that they do not have the same activation time. By launching R3, thus disconnecting M1, after that A2 discovers the address of M1, A2 does no more need to perform ARP poisoning. Indeed, disconnecting a machine is like inducing a denial of service on this machine. Consequently, A2 can directly spoof the address of M1 and fulfill its attack objective. Consequently, R3 has a side effect on the system, by increasing the risk of threat A2.

As you may notice, our framework is able to co-simulate the effects of each response on the system considering its activation time, allowing by this the system to choose the response plan bringing the highest risk mitigation.

6.2 Use Case 2

In Use case 2, we consider a system running a VoIP service, and a Trading service as shown in Fig. 7. For clients $\{cV1, cV2, ..., cV10\}$ subscribed to VoIP, a password based authentication using PAP (password authentication protocol) is considered and handled by a RADIUS server. While for clients $\{cT1, cT2, ..., cT8\}$

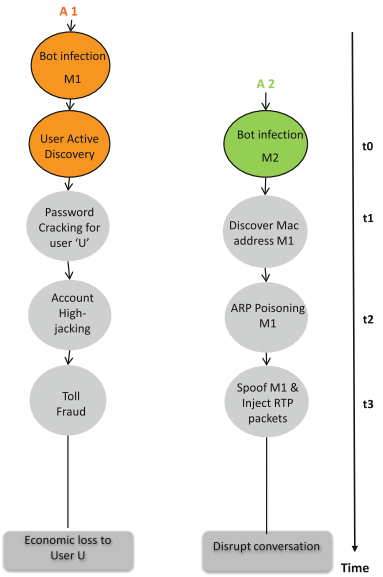

Fig. 3. SAG: System threatened by two attack entities A1 and A2.

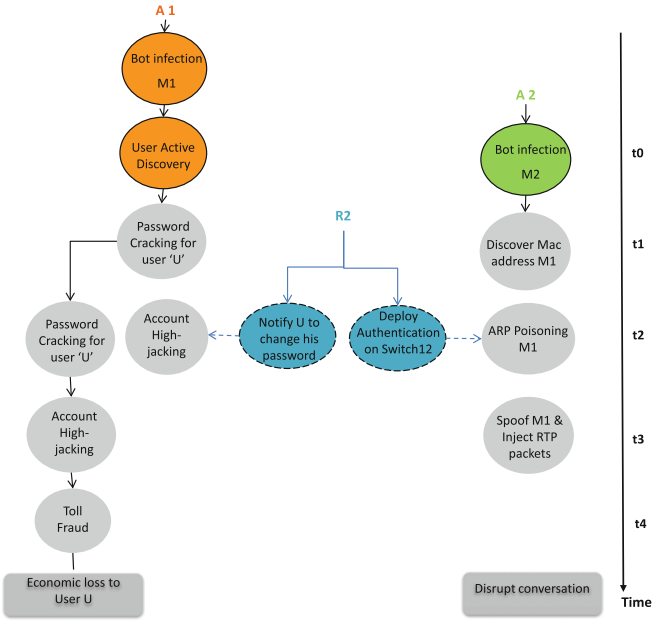

Fig. 4. Response against A1 and A2, designed as parallel actions.

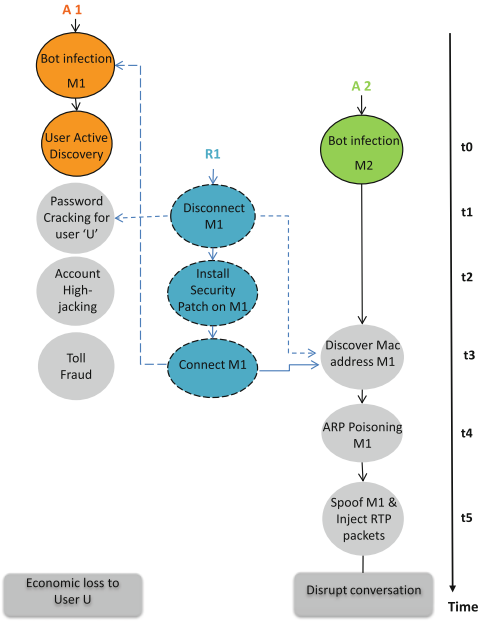

Fig. 5. Response against A1 and A2, designed as a sequence of actions.

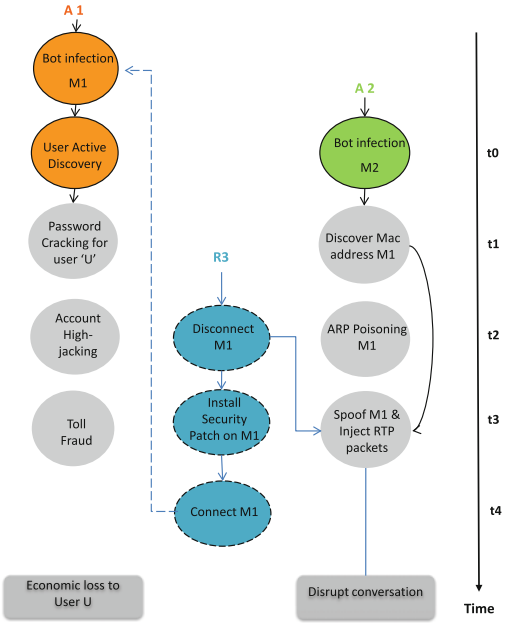

Fig. 6. Response against A1, having side effects on A2.

subscribed to Trading service, a strong authentication (e.g. multi-factor authentication, Digest access authentication, etc.) is adopted and handled by a Strong Authentication Server (SAS). SAS is suffering a Zero day vulnerability (e.g. Heart bleed[1]). Besides, in the initial system state, clients $\{cV1, ..., cV5\}$ and $cT1$ are compromised. The graph of Fig. 8 forecasts two different risky threats T1 and T2 led by these two attack entities. In T1, a coordinated password cracking attack scenario is predicted over $cV8$'s account. In T2, $cT1$ will try to exploit the vulnerability of SAS and prevent other traders from connecting to the trading service. The following is the attack sequence corresponding to the graph:

t1: [[*botinfect(a1,cV1),...,botinfect(a5,cV5), scanVulnerability(cT1,fw2)*];
t2: [*cscanuser*$((cV1, ..., cV5), cV8)$, *modifyAccessRules*$(cT1, fw2)$];
t3: [*cpassCrack*$((cV1, .., cV5), sV1, cV8)$, *scanserver*$(cT1, sas)$];
t4: [*highjack(cV1,sV1,cV8), exploitVulnerability(cT1,sas)*]]

In order to prevent T1, a solution would be to adapt the strong authentication to the VoIP service. To do so, the database containing information (passwords, accounts, etc.) about VoIP clients should be transferred to server SAS. Thus, r_1 =*transferData(sV1,SAS)* is anticorrelated with *cpassCrack((cV1,...,cV5),sV1,cV8)*. Another solution would be to notify $cV8$

[1] http://www.zdnet.com/heartbleed-serious-openssl-zero-day-vulnerability-revealed-7000028166/.

to change his password before that $cV1$ highjacks his account. Thus, action $r_3 = changePassword(sV1, cV8)$ is anticorrelated with $highjack(cV1, cV8, sV1)$.

In order to prevent T2, a solution would be to disconnect SAS in order to install security patches or a new software version (e.g. OpenSSL 1.0.1g) to patch the vulnerability. Thus, action $r_2 = installPatch(sas)$ is anticorrelated with $exploitVulnerability(cT1, sas)$. Another solution would be to discard or blacklist the malicious trader for a while. Thus, $r_4 = discard(cT1)$ is anticorrelated with all actions executed by $cT1$.

A naive solution to respond to both threats would be to choose any combination $[r_i, r_j]$, with i an even number and j an odd number. However, r_1 and r_2 are in conflict. Actually, installing the security patch requires disconnecting sas from the network, whereas transfering data to sas requires this latter to stay online. Consequently, our framework prevents the execution of these two actions in parallel, by appealing the notion of resource: $xres(r_2, sas) \wedge sres(r_1, sas)$. Thus, $conflict([r_1, r_2])$ returns true, and $\boldsymbol{Poss}([r_1, r_2], S)$ returns false. Our planner avoids, thus, conflicting actions while designing the response plans:

Response Plan 1 presented in Fig. 9 integrates r3 and r2:

t1: [[*botinfect(a1,cV1),..,botinfect(a5,cV5), scanVulnerability(cT1,fw2),* ***diconnect(sas)***];
t2: [*cscanuser((cV1,..,cV5),cV8), modifyAccessRules(cT1,fw2),* ***installPatch(sas)***];
t3: [*cpassCrack((cV1,..,cV5),sV1,cV8),* ***restart(sas)***];
t4: [***changePassword(cV1,cV8)***, *scanserver(cT1,sas)*];
t5: [*cpassCrack((cV1,..,cV5),sV1,cV8)*];
t6: [*highjack(cV1,cV8,sV1)*]]

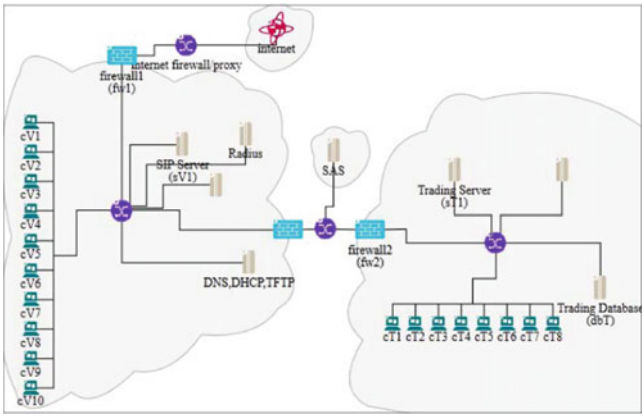

Fig. 7. Multi-services system topology.

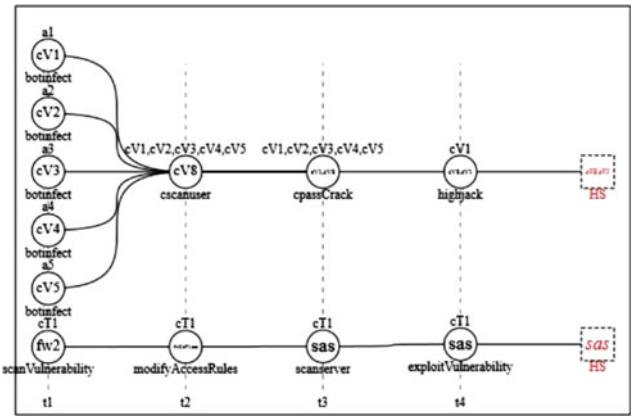

Fig. 8. SAG for the multi-services system.

Response Plan 2 presented in Fig. 10 integrates r1 and r4:

t1: $[[botinfect(a1,cV1),...,botinfect(a5,cV5),scanVulnerability(cT1,fw2)];$
t2: $[cscanuser((cV1,...,cV5),cV8),$ ***transferData(sV1,sas)***, ***discard(cT1)***$]]$

7 Related Work

Stakhanova et al. [17] proposed a response selection mechanism that can be based on a (i) static mapping, (ii) dynamic mapping, or (iii) cost-sensitive mapping. As opposed to static mapping, in dynamic mapping, the countermeasure is determined in realtime by considering additional factors related to the attack occurrence (e.g. attack confidence, attack severity, past experience). Cost-sensitive response systems can be viewed as a particular form of dynamic mapping. In such response systems, the selection procedure considers mainly the impact of the attack on the monitored system, and the cost of candidate countermeasures.

Kanoun et al. [9] highlighted the lack in existing taxonomies of considering the deactivation phase of a response. They, proposed a novel temporal response taxonomy using the Set Theory. Their taxonomy addresses the lifetime and the deactivation aspects of response measures distinguishing two major classes of countermeasures: one-shot and sustainable. Thus, response measures can be classified with respect to their effectiveness, lifetime, defeasibility, etc.

Unfortunately, most of the existing response taxonomies are based on a matching between a threat and a predefined response. Hence, an expert is needed to, first, understand and reason about each threat, and then, specify the response policy, in advance, for every threat. Besides, the potential conflicts between simultaneous responses, and the potential side effect of responses on the system, are not considered.

In [3], different types of conflict between responses are described, and a solution to avoid the conflict was proposed. This latter consists in performing, offline,

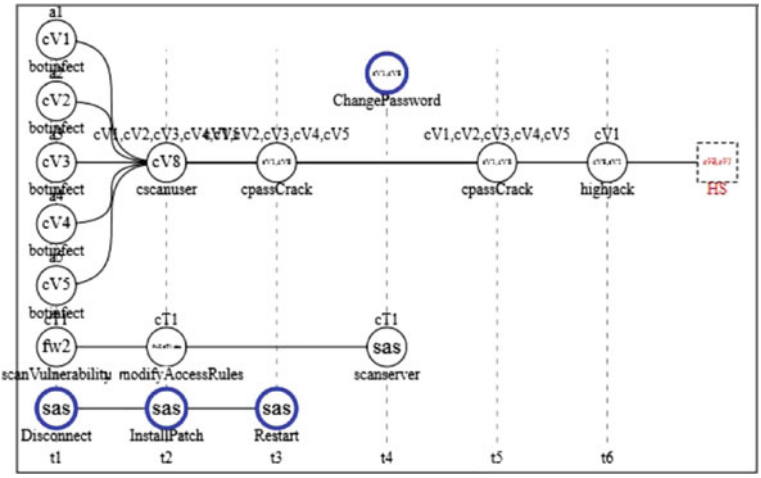

Fig. 9. Response Plan 1.

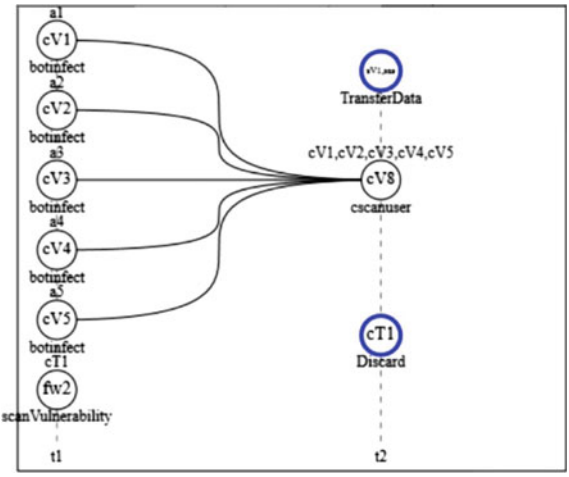

Fig. 10. Response Plan 2.

a static assignment of priorities over conflicting responses. However, conflicts between responses may depend on the current system's state and the dynamic resources' allocation. Thus, the conflict should be dynamically handled.

In [6], authors introduced a structured approach to evaluate a Return On Response Investment (RORI) index for all possible combinations of security measures that can be launched against simultaneous threats. In this work, security measures corresponding to each threat are designed by an expert in advance. Moreover, the risk mitigation for combined countermeasures is calculated by adding the effectiveness of countermeasures over the different surfaces they cover.

An attack surface is defined as the subset of the system's resources that an attacker may use to send/receive data into/from the system in order to attack the system. Thus, the effectiveness of a combination of responses, and thus its risk mitigation, is restricted to the threats for which these responses are considered. However, a proper risk mitigation should be calculated over the totality of the ongoing threats including not yet risky ones.

8 Conclusion

In this paper, we proposed a new response scheme for simultaneous threats, as a sequence of non conflicting parallel actions. Our response is dynamically designed based on a new definition of capability-aware logic anticorrelation, and modeled using the Situation Calculus language. This latter is efficient to describe conflicts between parallel actions by appealing the notion of resource. Moreover, in order to choose the most effective response, when multiple responses are possible, we presented a co-simulator based on SC planning capabilities. This latter co-simulates each response possibility apart, considering the system's state and the currently existing attack entities. Our framework is implemented in SWI-prolog, and experimentations were led to reveal the benefits of our solution.

In the future, we intend to assess the risk mitigation and the return on investment of each response plan in order to activate the most efficient one.

References

1. Boutilier, C., Brafman, R.I.: Partial-order planning with concurrent interacting actions. J. Artif. Int. Res. **14**(1), 105–136 (2001)
2. Cuppens, F., Autrel, F., Bouzida, Y., Garcia, J., Gombault, S., Sans, T.: Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection network (2006)
3. Cuppens, F., Cuppens-Boulahia, N., Bouzida, Y., Kanoun, W., Croissant, A.: Expression and deployment of reaction policies. In: IEEE International Conference on Signal Image Technology and Internet Based Systems, SITIS 2008, pp. 118–127, November 2008
4. Cuppens, F., Ortalo, R.: LAMBDA: a language to model a database for detection of attacks. In: Debar, H., Mé, L., Wu, S.F. (eds.) RAID 2000. LNCS, vol. 1907, pp. 197–216. Springer, Heidelberg (2000)
5. Essaouini, N., Cuppens, F., Cuppens-Boulahia, N., Abou El Kalam, A.: Specifying and enforcing constraints in dynamic access control policies. In: 2014 Twelfth Annual International Conference on Privacy, Security and Trust (PST), pp. 290–297. IEEE (2014)
6. Gonzalez Granadillo, G., Belhaouane, M., Debar, H., Jacob, G.: Rori-based countermeasure selection using the OrBAC formalism. Int. J. Inf. Secur. **13**(1), 63–79 (2014)
7. Irvine, C., Levin, T.: Toward a taxonomy and costing method for security services. In: Proceedings of the 15th Annual Computer Security Applications Conference, ACSAC 1999, pp. 183–188. IEEE Computer Society, Washington, DC (1999)

8. Jr., C.C., Pooch, U.W.: An intrusion response taxonomy and its role in automatic intrusion response. In: The 2000 IEEE Workshop on Information Assurance and Security (2000)
9. Kanoun, W., Samarji, L., Cuppens-Boulahia, N., Dubus, S., Cuppens, F.: Towards a temporal response taxonomy. In: Di Pietro, R., Herranz, J., Damiani, E., State, R. (eds.) DPM 2012 and SETOP 2012. LNCS, vol. 7731, pp. 318–331. Springer, Heidelberg (2013)
10. Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: Golog: a logic programming language for dynamic domains (1994)
11. Mccarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In: Machine Intelligence, vol. 4 (1969)
12. Pinto, J.A.: Temporal reasoning in the situation calculus (1994)
13. Reiter, R.: Natural actions, concurrency and continuous time in the situation calculus. In: Aiello, L.C., Doyle, J., Shapiro, S.C. (eds.) KR, pp. 2–13. Morgan Kaufmann, San Francisco (1996)
14. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press, Massachusetts, Illustrated edition (2001)
15. Samarji, L., Cuppens, F., Cuppens-Boulahia, N., Kanoun, W., Dubus, S.: Situation calculus and graph based defensive modeling of simultaneous attacks. In: Wang, G., Ray, I., Feng, D., Rajarajan, M. (eds.) CSS 2013. LNCS, vol. 8300, pp. 132–150. Springer, Heidelberg (2013)
16. Samarji, L., Cuppens-Boulahia, N., Cuppens, F., Kanoun, W., Papillon, S., Dubus, S.: Liccas: assessing the likelihood of individual, coordinated, and concurrent attack scenarios. In: Security and Privacy in Communication Networks (2014)
17. Stakhanova, N., Basu, S., Wong, J.: A cost-sensitive model for preemptive intrusion response systems. In: Proceedings of the 21st International Conference on Advanced Networking and Applications, AINA 2007, pp. 428–435. IEEE Computer Society, Washington, DC (2007)
18. Templeton, S.J., Levitt, K.: A requires/provides model for computer attacks. In: Proceedings of the 2000 workshop on New security paradigms, NSPW 2000, pp. 31–38. ACM, New York (2000)
19. Wang, H., Wang, G., Lan, Y., Wang, K., Liu, D.: A new automatic intrusion response taxonomy and its application. In: Shen, H.T., Li, J., Li, M., Ni, J., Wang, W. (eds.) APWeb Workshops 2006. LNCS, vol. 3842, pp. 999–1003. Springer, Heidelberg (2006)
20. Zhou, C.V., Leckie, C., Karunasekera, S.: A survey of coordinated attacks and collaborative intrusion detection. Comput. Secur. **29**(1), 124–140 (2010)