Multiple-Sites Defense Strategy

Elie Bursztein
LSV, ENS Cachan, CNRS, INRIA, France
eb@lsv.ens-cachan.fr

Abstract

The anticipation game framework is an extension of attack graphs based on game theory. This framework is used to analyze network incident scenarios, such as intrusion, and to find the best strategies associated to it. In this paper we present an extension to anticipation games to deal with multiple sites analysis. This extension is needed to find strategies that involve inter-sites communication and to model scenarios where a discrete timeline of events is required. To illustrate this extension, we present how multiple-sites defense can be used to mitigate a zero day attack. We prove that this extension is decidable and does not change the anticipation game complexity. Finally we evaluate the ability of anticipation game to analyze complex multiple-site defense with our implementation called NetQi.

1 Introduction

As networks of hosts continue to grow, evaluating their vulnerability to attacks becomes increasingly more important. When evaluating the security of a network, it is not enough to consider the presence or the absence of public vulnerabilities. Inevitably, a large network will contain undisclosed vulnerabilities that can be the target of undisclosed attacks called zero day exploits. To setup a defense strategy that mitigates this kind of attack, multiple-sites defense needs to be considered.
Using network topological information along with other information such as average deployment cost and time, an analyst can produce an **anticipation game**.The anticipation game framework [7] is an extension of attack graphs based on game theory [14] and more specifically on TATL [15] (Timed Alternating-time Temporal Logic). Using a timed game allows to model player actions on the network. An anticipation game is described by an initial state and one set of timed rules for each player. Instead of using a state reachability property as analysis goal, it uses strategy objectives. Strategy objectives permit the expression of analysis goals such as: what is the most efficient patching strategy. It can be intuited as finding every execution that leads to the desired result and then selecting among them the one that fulfills the best player objectives in term of cost, reward, and time. This is possible because anticipation games take into account the notion of action cost, time, and reward. Rewards are based on network service value. Therefore it is also possible to define strategy objectives that will find the worst attack against the network.

1.1 Motivation

Anticipation games can serve as a basis for attack anticipation, defense strategy analysis and forensic analysis. To motivate our extension of the framework we discuss the potential applications of this extension. zero day attack cannot be mitigated by the use of an intrusion detection system based on misuse detection, or patching because they are by definition undisclosed. A way to be aware that such undisclosed attacks occur is to rely on other sites to be alerted when they are hit by an unknown attack. A common security practice is to use a dedicated trap net called a honey-net for this purpose. A honey-net will issue an alert when an unknown attack is used against one of its services. This alert will then be used by real sites to take counter-measures, such as denying the intruder's IP in their firewalls.

Modeling a multiple-sites strategy cannot be done in attack graphs and the current anticipation games framework, because some rules and strategies need to be restrained to specific set of services, whereas other need to be global to model inter-sites communication. For example, patching rules should not be applied to honey-net network. That is why location restriction needs to be introduced in the framework. Moreover dealing with a zero day exploit requires taking into account the vulnerability cycle timeline. This is not possible without the location restriction. Mass-scale attacks such as worms can also be detected through sites cooperation. Without our extension they cannot be modeled in anticipation games either because

they require a timeline of events and rules restriction as well. Theses two applications also require the modeling of the notion of cost by unit of time. In the zero day attack case, monitoring a honey-net costs money on a daily basis. In the mass-scale attack case, service unavailability causes a loss of income based on the duration of the unavailability. To model this type of cost, we extend anticipation games with penalties. Intuitively a penalty is a cost added for each unit of time a constraint holds. This extension also allows us to model an other important relation between time and cost: cost diminishing. A cost diminishing occurs when the same action is executed multiple times. It also occurs when an on-going process is done for an extended period. *e.g* service monitoring.

Finally this extension allows us to model player's simultaneous actions and event branching. Simultaneous actions are mandatory to model that every site takes the same counter-measure simultaneously when a zero day attack is detected by the honey-net. It can also be used to model the massive deployment of a patch. Event branching is used to model that at a certain point, several options are available to the player, such as using one kind of patch or an other.

1.2 Our contribution

The main contribution of this paper is an extension of the anticipation game framework to model multiple-sites interaction. This extension is three fold.

First, locations are introduced in order to control the scope of rules and strategies. Secondly the use of a discrete timeline of events is introduced to model causal relations between events. For example, a discrete timeline of events can be used to model that a vulnerability is reverse engineered `only after` it has been caught on a honey pot. Thirdly we extend the framework to model the relation that exists between cost and time. We add the notion of penalty to model costs that are time dependent and costs that diminish over time.

To illustrate how this extension is used to model multiple-sites interaction, the running example presented in this paper discusses a multiples-sites defense strategy that uses a honey-net against various types of exploits including zero day ones.

We prove that anticipation games with locations and penalties are decidable and that the complexity of the model remains EXPTIME-complete. This extended anticipation games framework has been implemented into an freely available tool called NetQi [6] to evaluate the effectiveness of the approach. In the evaluation section we show that it is possible to analyze complex multiple-sites scenarios.

The reminder of this paper is organized as follows. In Sect. 2, we will survey related work and in Sect. 3 we recall what an anticipation game is. We also detail the game example that is used as a guideline for the rest of the paper. Sect. 4 presents the notion of locations. In Sect 5 the discrete timeline of events is illustrated. Sect. 6 introduces the notion of penalty and cost diminishing. Sect 7 covers the multiples-sites defense strategies that were found by analyzing the running example with NetQi. In sect. 8 we evaluate NetQi performance. We conclude in Sect. 9

2 Related Work

Attack graphs are a very active field pioneered by Schneier [32, 33] and Kuang and al [41]. Model checking for attack graphs was introduced by Ammann and Ritchey [30]. They are used to harden security [25]. Various methods have been proposed for finding attack paths, i.e., sequences of exploit state transitions, including logic-based approaches [28, 36, 17, 35], and graph-based approaches [41, 37, 24]. Researchs have also been conducted on formal languages to describe actions and states in attack graphs [12, 38]. Some rely on grammars [39], some have a more practical focus [11], or specialize on IDS alert correlation [23]. Some authors propose techniques that allow attack graphs to scale to large networks [16, **?**]. Security metrics [26] have been developed, and supporting tools such as NetSPA [3] or Sheyner's tool [36] now exist.

The SIR model, which is similar to the compromising recovery cycle, is used to study the propagation of epidemics in biology [10]. Biological models for computer security were proposed recently [31]. As in computer virus propagation research [5, 40], biological models are an inspiration of anticipation games. The antibody (administrator) fights the disease (Intruder) to maintain the body alive (the network). Following this intuition, using games to capture this fight interaction appears natural.

Games have become a central modeling paradigm in computer science. In synthesis and control, it is natural to view a system and its environment as players of a game that pursue different objectives [9, 27]. In our model, the intruder attempts at causing the greatest impact on the network whereas the administrator tries to reduce it. Such a game proceeds for an infinite sequence of rounds. At each round, the players choose actions to play, e.g., patching a service, and the chosen actions determine the successor state. For our anticipation games we need, as in any *real-time* system, to use games where time elapses between actions [22]. This is the basis of the work on timed automata, timed games, and timed alternating-time tempo-

ral logic (TATL) [15], a timed extension to alternating-time Kripke structures and temporal logic (ATL) [2]. The TATL framework was specifically introduced in [14].

Timed games differ from their untimed counterpart in two essential ways. First, players have to be prevented from winning by stopping time. More important to us is that players can take each other by *surprise*: imagine that the administrator attempts to patch a vulnerable service, and this will take 5 minutes, it may happen that intruder is in fact currently conducting an attack, which will succeed in 5 seconds, nullifying administrator action. Second (this is allegedly more technical), a player cannot win by preventing time from diverging, i.e., from eventually tending to infinity [34]. Average reward games considered in TATL framework are considered in [1], but with the time move duration restricted to either 0 or 1. ATL was also extended to simply timed concurrent game structure [18]. Game strategies have been used to predict players actions in numerous domains ranging from economy to war [4, 29].

The notion of cost diminishing appears in [13]. The use of games for network security was introduced by Lye and Wing [20]. The anticipation game framework was presented by Bursztein and Goubault-Larrecq [7] and network strategies for anticipation games were detailed in [8]. Finally the first use of game theory for denial of service was done by Mahimkar and Shmatikov [21].

3 Anticipation Games

A key difference between standard timed games [14] and anticipation games is the dual-layer structure used in anticipation games. Anticipation games lower-layer is called **Network Layer**, and is used to represent network information. The upper-layer is called **Attack Layer** and is a regular TATL game structure used to model the network state evolution induced by players actions. Anticipation games can be intuited as a graph of graphs (see diagram 1) where the lower graph is the network state and the above graph describes the transition between one network state to an other.

Anticipation games players are called administrator and intruder and their actions are modeled by **timed rules**.Typical actions range from patching, to exploiting a vulnerability, to firewalling a service. They are called timed rules because a rule execution requires a certain amount of time to be executed. Each Attack Layer transitions represent the execution of one rule. An anticipation games path is called a **play**. More formally a play is a path (a sequence of action and states) $\rho : s_0 r_0 s_1 r_1 ...$ where $\forall j : s_j \xrightarrow{r_j} s_{j+1}$, s_j and s_{j+1} are network states, and r_j is the rule used to make the transition.

The main purpose of anticipation games with strategies is to find, given a network initial state and a set of rules, the best strategy that fulfills player objectives by model-checking the Attack Layer and comparing the outcome of each play that fulfills strategy constraints. Hence a player **strategy** is the play that satisfies the most player strategy objectives in term of cost, reward and time.

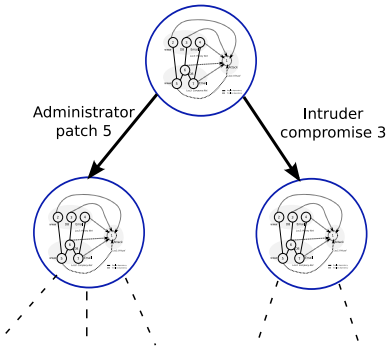

Figure 1. Anticipation games intuitive representation

3.1 Network Layer

The Network Layer is composed of two parts. First the **Dependency Graph** which is the graph that represents the dependency relations that exist in the network. It is meant to be static and does not evolve over game execution. Secondly a finite **set of states** associated to each Dependency Graph vertex that describes the current network state. This set of states is meant to evolve over game execution. Typical states range from vertex vulnerability, to vertex public accessibility, to vertex compromising. A state is a Boolean value. More formally, let $\mathcal{A}$ be a finite set of so-called *atomic propositions* $A_1, \ldots, A_n, \ldots$, denoting each base property. Thus each atomic proposition is true or false for each of Dependency Graph vertices.

The Dependency Graph used as an example is represented in diagram 2 and the corresponding set of states is represented in the array 3. This Dependency Graph uses the concepts of locations and of discrete timeline of events introduced later in the paper.

3.2 Dependency Graph

This Dependency Graph is composed of six real vertices and a virtual one (vertex 1). The edges are the dependencies that exist between services. Concrete dependencies are represented with a plain line and virtual dependencies used

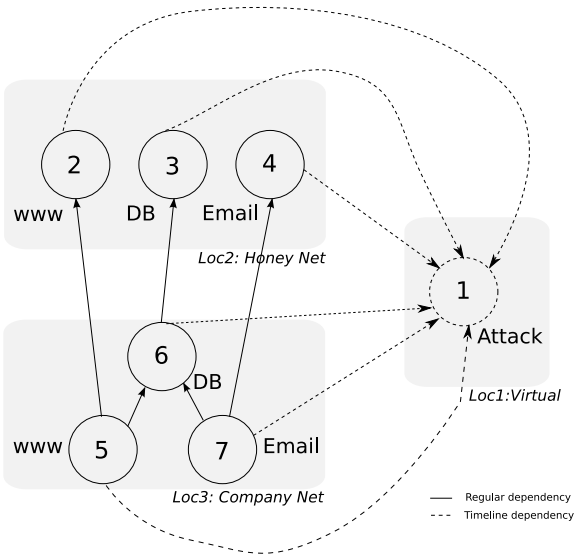

Figure 2. Dependency Graph

for the discrete timeline of events with a dashed line. The role of the virtual vertex and its incoming dependencies is to model the discrete timeline of events as detailed in section 5. Concrete dependencies are used to model that a service is dependent on another. In our example the vertex *www (5)* depends on the vertex *DB (6)* for user authentication. From a security perspective it means that if the vertex *DB (6)* is unavailable by **collateral effect** the vertex *www (5)* will be also unavailable. It also means that the trust relation that exists between those two vertices may be exploited by an attacker. These dependencies are used in game rules to model collateral effects and trust abuse. The three dependencies from the company's network services to their twins services located in the honey network are used for the multiple-sites defense purpose. The company's services depend on honey-net fake services to defeat a zero day attack as exemplified in section 7.

3.3 Set of states

The complete set of states mapping used in the example can be divided into three parts. The first part is sets `0DayAvail`, `CustomAvail`, `PubAvail` and `PatchAvail` which are used to model the discrete timeline of events as detailed in section 5. The second part is sets `Detected` and `Monitored` which are used for multiple-sites defense purpose (see section 7). And finally the third part is used to describe the network original state. More specifically the set `Vuln` is used to say that the company DB service and its fake twin service on the honey-net are vulnerable to an unknown vulnerability at the beginning of the game. The set `Compr` is used to say that no service is compromised at the beginning of the game. Finally the set `Public` is used to indicate that every service is public (not firewalled).

States	1	2	3	4	5	6	7
ρ(0DayAvail)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(CustomAvail)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(PubAvail)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(PatchAvail)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(Detected)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(Monitored)	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
ρ(Vuln)	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
ρ(Compr)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(Public)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Figure 3. Set of states

3.4 Attack Layer

TATL [15] extends ATL [2] with the notion of timed game. This is done by adding time cost to transitions. From the network security perspective this is important because it means that player actions on a network require a certain amount of time to be executed. This prevents meaningless strategies such as being able to patch every network vulnerability in an instant. Hence an anticipation game can be viewed as a race between players where the fastest wins. As presented in [14], this time race introduces in the game the element of surprise. For example the intruder can take the administrator by surprise if he can exploit a vulnerability faster that the administrator can patch it. This is coherent with real network security where you cannot foresee what attacker will come up next.

3.5 Rules of the game

Legal actions for each player are described by a set of timed rules. Each rule is of the form:

$$\Gamma_x : \mathbf{Pre}\ F \xrightarrow{\Delta,\ \mathrm{p},\ \mathrm{a},\ \mathrm{c}} P$$

where F is the set of **preconditions** that need to be satisfied in order to use the rule. Δ is the amount of time needed to execute the rule, p is the player that uses the rule, a is the rule label (string), and c is the rule cost. P is the rule **post-condition**, that states rule effects. It is required for F preconditions to hold not just when the rule is selected, but also during the whole time it takes the rule to actually complete (Δ time units). Γ_x is the rule location. Anticipation games use two types of rules [8]. The **granting rule**

that uses the $\Longrightarrow$ double arrow and the **regular rule** that uses the $\longrightarrow$ single arrow. A granting rule allows the player to receive a reward based on the target Dependency Graph vertex value when the rule is successfully executed whereas regular rule does not grant any reward. Regular rules are used for temporary actions and for the discrete timeline of events. For example the following rule is used to model trust abuse attack:

$$\Gamma : \mathbf{Pre}\ \Diamond Compr \wedge \neg Compr \overset{\text{2,I,Trust abuse, 200}}{\Longrightarrow} Compr$$

It says that the intruder (I) can compromise a non compromised ($\neg Compr$) vertex by exploiting a trust relation if one of its successors is compromised ($\Diamond Compr$) in 2 units of time for a cost of \$200. The $\Diamond$ is a modal operator used to speak of Dependency Graph successors. The other operators used in rule preconditions and effects are standard S4 operators. If the intruder chooses to use this rule, then to have a successful rule execution it is required that the preconditions are fulfilled when he chooses to apply the rule, and also after the 2 units of time required to complete it. This is mandatory because the network state might evolve due to administrator actions during these 2 units of time. For example the administrator might clean the successor vertex. In this case, the intruder is taken by surprise, and the compromise rule fails.

4 Location

In anticipation games a rule can be applied to any Dependency Graph vertex as long as its set of states conditions meets the rule preconditions requirements. However in many cases such behavior is not suitable. In particular it is not possible to perform multiple-sites defense analysis without restricting the scope of rules. This impossibility is mainly due to the fact that for this type of analysis three type of rules are needed the **transitive** ones, the **local** ones, and the **global** ones. Transitive rules are used to model inter-site interaction. Local rules are used for site specific action. In our example the rule used to model trust abuse attacks needs to be restricted to company's network, and discrete timeline of events rules to the virtual location. Finally global rules are meant to be used on any vertex. Similarly strategies objectives need to be restricted to a given set of vertices. In our example : finding a defense strategy that prevents service compromising should obviously not apply to honey-net fake services.

In order to restrict rules and strategy objectives to a given set of vertices, we extend anticipation games with locations. Intuitively a location is a group of services that belongs to the same site. More formally a location is a set of Dependency Graph vertices represented by an integer. Location integer is added to every Dependency Graph vertex as a label. Locations are specified in rules and strategy objectives to restrict their scopes.

4.1 Type of Rule

We use the set of an **operational rules** depicted in figure 4 in the example. We speak of operational set because it is used to model attack and defense actions. At the opposite the set of **timeline rules** depicted in section 5 is used to model timeline events. This operational set combines the three types of rules to model multiple site defense. The three type of rules are more formally defined as:

Definition 1. (Global rule) *A rule is global if no location restriction is specified.*

Definition 2. (Local rule) *A rule is local if the same location restriction is specified for the rule target vertex and the rule target successor vertex.*

Definition 3. (Transitive rule) *A rule is transitive if a different location restriction is specified for the rule target vertex and the rule target successor vertex.*

4.2 Global Rules

The first three rules are comparable, as they model the same action: an intruder (I) that exploits a remote service vulnerability to compromise a public service. The rule preconditions ensure that the target service is vulnerable ($Vuln$ has to be true) and remotely accessible ($Public$ has to be true). The rule effects when the execution is successful is that the vertex becomes compromised ($Compr$ become true). Since these rules are meant to attack fake and real services they are global (Γ has no index). They differ because due to the events timeline, they are available at a different time. For the rest of the paper we use the standard anticipation games modal operator $\Diamond$ to speak of regular dependency and $\Diamond_{\gg}$ to speak of a timeline dependency. They both denote the same anticipation games modal operator $\Diamond$ and this distinction is only made to improve rule readability. The 0Day exploit is released first, then the custom exploit and finally the public exploit. For instance $0DayAvail$ is set to true for the virtual vertex by a timeline rule after 48 hours. This set is used to prevent the intruder from using it earlier in the game. Accordingly the Custom exploit cannot be used before it is available because until then, the $CustomAvail$ is set to false for the virtual vertex. The cost of the three rules also differs from modeling that researching a vulnerability is more costly than making a custom exploit which is more costly than simply using a public exploit. The conjunction of cost and timeline allows to model

1) $\Gamma_{:}$: **Pre** : $\Diamond_{\gg} 0DayAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 3, I, 0 day exploit, 20000
Effect : $Compr$

2) $\Gamma_{:}$: **Pre** : $\Diamond_{\gg} CustomAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 4, I, Custom exploit, 2000
Effect : $Compr$

3) $\Gamma_{:}$: **Pre** : $\Diamond_{\gg} PubAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 7, I, Public exploit, 200
Effect : $Compr$

4) $\Gamma_{3:3}$: **Pre** : $\neg Compr \wedge \Diamond Compr$
$\Longrightarrow$ 2, I, Trust Abuse, 200
Effect : $Compr$

5) $\Gamma_{1:1}$: **Pre** $Monitored \wedge Compr \wedge \neg Detected$
$\longrightarrow$ 1, A, Attack Detected, 2000
Effect $Detected$

6) $\Gamma_{2:2}$: **Pre** $\neg Vuln \wedge \neg Public$
$\longrightarrow$ 1, A, Unfirewall, 100
Effect $Public$

7) $\Gamma_{3:2}$: **Pre** $\Diamond Detected \wedge Vuln \wedge Public$
$\longrightarrow$ 0, A, Firewall, 100
Effect $Public$

8) $\Gamma_{3:1}$: **Pre** $\Diamond_{\gg} PatchAvail \wedge Vuln$
$\longrightarrow$ 6, A, Patch, 500
Effect $\neg Vuln$

Figure 4. Set of rules used to model a players action

the trade-off between the advantage awarded by an undisclosed vulnerability exploit and the investment required to find it.

4.3 Local Rules

The rules 4, 5, and 6, are local rules. Their Γ index is of the form $n : n$ where the first n is the vertex location and the second n is the successor location. The rule 4 says that if a service is not compromised ($\neg Compr$) and if one of its successor is compromised ($\Diamond Compr$) then it can be compromised by the intruder (I) in 2 hours for $200. This rule must be local because otherwise erroneous actions are possible. As visible in the diagram 2 a dependency exists between each company's service and its corresponding honey-net service. When the trust abuse rule is not restricted to a local scope these relations can be used for trust abuse. As a result a compromised honey-net service can be used to compromise a company's service by trust abuse, which is clearly an erroneous action. That is why this rule needs to be restricted to the company's network context to be executed only on services where real trust relation exists.
The rule 5 is local to the honey-net network. It states that if a service is monitored ($Monitored$), compromised ($Compr$) and an alert has not been already raised ($\neg Detected$) then an alert is raised. The time required to trigger the rule also includes the alert propagation time in order to achieve simultaneous service firewalling execution as explained in section 5. The $Monitored$ set is used as detailed in section 6 to compute monitoring ongoing process cost.
The rule 6 is local to the company's network because since the firewall rule applies only to the company's network this one should only apply to it as well. It states that if a service is not public ($\neg Public$) and not vulnerable ($\neg Vuln$) then it can be made public ($Public$).

4.4 Transitive Rules

Rules 7 and 8 are transitive rules. Their Γ index is of the form $n : m$ where n is the vertex location and m the successor location. They are used for multiple-sites interaction. In the example there are two kinds of such interactions. First the interaction between the honey-net (location 2) and the company's network (location 3). This interaction allows the company's network to defend itself against unknown attacks by firewalling a company's service when the corresponding honey-net service experience an attack. This interaction is described by the rule 7 which states that if an attack is detected on a remote location ($\Diamond Detected$) and the vertex is public (*Public*) and vulnerable (*Vuln*) then it can be firewalled by the administrator. The location restriction ensures that only company's network will be affected by the rule. It also ensures that the successor belongs to the honey-net.
The other transitive rule is the patching rule. It is restricted to the company network location because honey-net services are not meant to be patched. Its successor has to be the virtual location because this is where the discrete timeline of events evolves. The timeline information is needed to know when the patch is available. This rule can only be transitive: if it is global, it can be applied to honey-net and if it is local it does not work because the discrete timeline of events evolution take place in the virtual location.

4.5 Strategy with location

The strategy tuple is extended with a 5th optional component used to add strategy locations constraints. These constraints allow us to restrict or exclude locations from strategy scope. For instance, excluding a location is used in the example to find a defense strategy for every sites except the honey-net. These defense strategy objectives are defined as:

$$S : (\text{Defense strategy}, Admin, MIN(Cost) \wedge MAX(OCost), OCost > Cost, \Box\neg Compr, \neg 2)$$

They are used to find the play that maximizes intruder cost, minimizes the administrator cost and ensures that no service in every location except the honey-net location ($\neg 2$) is ever compromised ($\Box$). Adding the opponent cost maximization objective aims at finding the **(weakly) dominant strategy**: The strategy that beats every opponent strategy (strict dominance) or at least maximizes the number of strategies beaten (weak dominance). It can be intuited as finding the play where the opponent plays his best game. Since the number of locations is finite it is possible to replace this strategy by n strategies where n is the number of locations to consider. However this is less readable and having a single strategy leads to serious performance improvement, due to early cuts, as detailed in section 8.

Lemma 1. *Anticipation games extended with locations are decidable.*

Proof. Suppose that for each location λ_x, a set of states σ_x is added. Locations are used in three game components: dependency graph vertex, rule and strategy. Each dependency graph vertex is bounded to a given location. This binding can be done by encoding location in σ_n sets where n is the number of location present in the game. The encoding is done by initializing the set $sigma_x/x < n$ to true for each vertex that belongs to the location λ_x and false otherwise. In a rule a location restriction can be applied to target node and target successor. The node restriction to location λ_x can be enforced with $sigma_n$ sets by adding to the preconditions the extra precondition that σ_x is true for local and transitive rules. This extra precondition can be added because the lower layer modal logic allows the use of the standard conjunction operator. Similarly successor location restriction to location λ_x can be enforced with σ_n set by adding in preconditions the extra precondition $\Diamond\sigma_x$ for local and transitive rules. Restricting a strategy to the location λ_x can be done by testing that $\rho(\lambda_x)$ is true for each node before verifying the strategy constraints. If it is false, the node is not considered. Hence, it is sufficient to add λ_n sets to a game to encode location. There are a finite number of locations because by definition there is a finite number of vertices in dependency graph, and a location contains at least one vertex. Therefore the number of set λ_n added is finite. By the theorem of [7] an anticipation game with a finite number of sets is decidable. It follows that anticipation game with locations is decidable. □

5 Using a Discreet Timeline of Events

Being able to model a discrete timeline of events is mandatory because many network security scenarios need it. For instance the classical vulnerability cycle [19] follows a discrete timeline of events: the patch for a given flaw is developed `only after` the vulnerability is either reported, or caught in the wild and reverse engineered. Similarly an attack can be detected by a misuse IDS `only after` its signature has been added to the database. Such a discrete timeline of events can be modeled in anticipation games by using a combination of rules, states and dependencies. The key idea is to add a virtual vertex in the dependency graph that is used to model the discrete timeline of events evolution thanks to a set of states. An additional set of dependencies from real services to this virtual vertex is added in order to be able to use discrete timeline of events state in rule preconditions and effects (as in the Dependency Graph depicted in figure 2). Locations are used to ensure that the virtual vertex is the only one used in discrete timeline of events evolution rules. Otherwise, every timeline rule will apply successively to every vertex leading to an erroneous strategy.

5.1 Discreet Timeline of Events Illustration

The multiple-site defense example uses a discrete timeline of events inspired by the standard vulnerability cycle represented in figure 5.

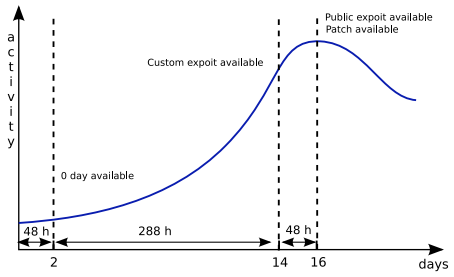

Figure 5. Vulnerability discrete timeline of events

To model this discrete timeline of events, four sets and four rules are needed (figure 6). Intuitively states are used to model which points have been reached so far and rules are used to advance in the timeline. One distinct state is required for each event because states are Boolean values. Accordingly each state used for the discrete timeline of events is set to false in initial conditions. The rule execution time represents the time interval between two consecutive events. For example the custom exploit is available 14 days after the vulnerability is discovered (global time), and 12 days after the zero day exploit (relative time). The availability of the custom exploit is modeled by the rule 2. This rule states that if the Custom exploit is not available ($\neg CustomAvail$) and the zero day is ($0DayAvail$) then after 288 units of time (12 days) the attacker will have access to custom exploit.

1) $\Gamma_{1:1}$: **Pre** $\neg 0DayAvail$
$\longrightarrow$ 48, I, O day exploit Available, 0
Effect $0dayAvail$

2) $\Gamma_{1:1}$: **Pre** $\neg CustomAvail \wedge 0DayAvail$
$\longrightarrow$ 288, I, Custom exploit available, 0
Effect $CustomAvail$

3) $\Gamma_{1:1}$: **Pre** $\neg PubAvail \wedge CustomAvail$
$\longrightarrow$ 48, I, Public exploit available, 0
Effect Pub

4) $\Gamma_{1:1}$: **Pre** $\neg PatchAvail \wedge CustomAvail$
$\longrightarrow$ 48, I, Patch available, 0
Effect $0dayAvail$

Figure 6. Set of rules used to model timeline evolution

5.2 Branching

Using a relative time allows us to model branching. For example if the timeline presented above is not sufficient, because one wants to model multiple ways to disclose the vulnerability and make the custom exploit available, then it is possible to use multiple rules that have the same effect but different preconditions, time, and cost. For example to model that the disclosure is the result of an intrusion caught by the honey-net and reverse-engineered the following rule can be used with the proper set of dependencies:

$\Gamma_{1:2}$: **Pre** $Detected$
$\longrightarrow$ 288, A, Reverse Engineering , 500
Effect $CustomAvail$

This transitive rule states that if a honey-net service is compromised then in 12 day the administrator staff is able to reverse engineer it for a cost of $500. Branching was not introduced in the example for the purpose of clarity.

5.3 Simultaneous Actions

Another type of discrete timeline of events occurs when multiple actions take place at the same time. In the multiple-site defense this occurs when the attack on the honey-net is caught: every site has to use the firewall simultaneously. Otherwise the time required to firewall x sites is equal to $x \times t$ where t is the time required to firewall one site. To have a constant time regardless of the number of sites a state is used as a validation point. In the example this is the state $Detected$. The time required to firewall the site is modeled by the rule 5 of figure 4. Once this rule is executed the administrator is able to use simultaneously as many firewall rules as she wants. This is achieved by setting the firewall rule time to 0.

6 Linking Cost and Time

In the original anticipation games model with strategies [8], costs are bounded to rule executions: each time a player executes a rule, his cost increases. This is a natural way to model that player action have a cost. However this approach has a limitation: it does not allow us to model costs that are time dependent. Such cost exists for on-going processes. Two well known examples of such on-going processes are service DOSing (Denial Of Service) [21], and intrusion detection monitoring. The longer they last, the higher the cost is. To model this type of cost, we extended anticipation games with penalty. Intuitively a penalty is a cost that is added for every unit of time a constraint holds on a given dependency graph vertex. More formally a penalty is defined as follows:

Definition 4. *(**Penalty**) A penalty is the tuple* $P : (\mathsf{P}, \mathsf{N}, \mathsf{C}, \mathsf{F})$ *where* P *is the player targeted by the penalty,* $\mathsf{N} \in \mathbb{N}^*$ *is the Dependency Graph vertex where the constraint has to hold,* C *is the constraint that needs to be satisfied to trigger the penalty, and* $\mathsf{F}(\mathsf{x})$ *is the function* $\mathsf{F}(\mathsf{x}) : \mathbb{N}^* \rightarrow \mathbb{N}$ *that takes as parameter the integer* x *which is the number of units of time elapsed since the penalty has been triggered and return the corresponding cost.*

6.1 Example

To make it clearer, let's illustrate how penalty can be used to model a DOS cost. Assume that the Dependency Graph vertex 5 is a HTTP service used to sell company products. Every hour, the amount of income generated by this service is $1000. Therefore for every unit of time the service is unavailable ($\neg$ *Avail*) because of the DOS, the company loses $1000 of income. This can be modeled by adding the following penalty to the game:

$$P : (\text{Administrator}, 5, \neg\text{Avail}, add(x) = 1000)$$

which states that for each unit of time where the vertex 5 (www service), is not available the administrator cost is increased by 1000. Thanks to this penalty the incident strategy cost minimization objective takes into account the relation between the loss of income and the time elapsed. In the example we use the same kind of penalty to compute the cost associated with the action of firewalling a public service. The use of a function based on the number of units of time elapsed allows us to model various cost models such as an exponential cost or a diminishing cost as presented above.

6.2 Cost diminishing

Another important time/cost relation to consider is when the cost diminishes over the time [13]. This reduction occurs when the same action is performed multiple times, or when an on-going process is run for an extended period of time.

Performing the same action again and again is a common practice in network security. For instance the action of patching similar services or reusing the same exploit. In this context the cost of the first use is more expensive than later ones. In the patching case, the first use is more expensive because it requires to download and test the patch. In the exploit case, the first use requires the attacker to develop and test the code whereas subsequent exploitations only require it to be launched

This type of cost reduction is modeled in anticipation games by using two rules with different costs and a discrete timeline of events to ensure that the cheaper rule is only used after the most expensive one has been used. It is very similar to modeling simultaneous actions except that it is cost reduction not time reduction.

Many network security devices such as intrusion detector and honey pot requires an on-going supervision process. This cost can be divided into two parts: the implementation cost and the supervision cost. Implementation cost covers every cost required to start the process ranging from software to hardware to setup and test. The supervision part covers every recurrent cost such as detection rules update and alert processing.

The implementation part is modeled in anticipation games by using an implementation rule whereas the supervision part is modeled by a penalty that is triggered by the successful execution of the implementation rule. To model a diminishing supervision cost with a lower limit the following penalty can be used:

$$P : (\text{Administrator}, 5, \text{Monitored}, cost : x \rightarrow int(1000/x)+y)$$

Where x is the number of time units elapsed, $y \in \mathbb{N}$ is the lower bound cost and $int(x)$ the standard function that returns a rounded integer from a float.

We use two kind of penalties in the example. The first kind is induced by monitoring honey-net service, we assume that monitoring a honey-net service costs \$10 by hour. Accordingly we add the three following penalties to the analysis:

$$P : (\text{Administrator}, 2, \text{Monitored}, add : 10)$$

$$P : (\text{Administrator}, 3, \text{Monitored}, add : 10)$$

$$P : (\text{Administrator}, 4, \text{Monitored}, add : 10)$$

The second kind adds penalty cost for each unit of time a company's service is not public because of the lack of income generated. For simplicity we assume that this loss is equal to \$1000 by unit of time for every service.

$$P : (\text{Administrator}, 5, \neg Public, add : 100)$$

$$P : (\text{Administrator}, 6, \neg Public, add : 100)$$

$$P : (\text{Administrator}, 7, \neg Public, add : 100)$$

Lemma 2. *Anticipation games extended with penalty are decidable.*

Proof. A penalty can either apply when a state is true or false for a given vertex. If two penalties exist for the same condition they can be merged into one by merging their cost function. Therefore there are at most $2 \times n \times m$ penalties where n is the number of Dependency Graph vertex and m is the number of set of states. Since there is a finite number of Dependency Graph nodes and a finite number of set of states by definition there is a finite number of penalty. The number of distinct states for any play is finite because the number of rules and Dependency Graph vertices are finite. Therefore an infinite play has a finite number of distinct states. It follows that if it is an infinite play, it has a loop. A penalty cost is monotonic, hence each time a penalty applies it strictly increases cost value. Consequently there are two possible cases for a given penalty. One: if the cost value is incremented by the penalty during the first loop iteration, it will be incremented at every loop iteration and therefore the cost value diverges. Two: the penalty does not increase the cost during the first loop iteration. In this case the cost will never be increased by penalty regardless of the number iterations. Therefore it is possible to compute the outcome of an infinite play with penalty in a short finite prefix. □

From lemma 1 and lemma 2 it follows that:

Theorem 1. *Anticipation games extended with penalties and locations are decidable*

Adding locations and penalties does not change the anticipation games complexity bound:

Theorem 2. *Model-checking TATL formula over anticipation games extended with penalties and locations remain EXPTIME-Complete*

Proof. Locations add at most two tests to each rule therefore testing locations add only a linear complexity factor to rule evaluation. Finding anticipation games strategies require to evaluate a finite number of states [8]. Or restricting strategy to a given set of locations requires to perform n tests for each evaluated state, where n is the number of Dependency Graph vertices. Therefore adding locations to a

strategy add $n \times s$ tests where n is the number of Dependency Graph vertices and s the number of states needed to decide an anticipation game. Therefore restricting strategies to a set of locations add only a linear complexity factor. It follows that anticipation games with location remains EXPTIME-complete.
The number of penalties is finite. Therefore penalties add at most $x \times s$ tests where x is the number of penalties and s the number of states. Thus penalties add only a linear complexity factor to anticipation games and therefore anticipation games remain EXPTIME-complete when extended with penalty. Location and penalty add both linear complexity factor therefore anticipation games extended with location and penalty remain EXPTIME-complete. □

7 Multiple sites Strategies

We use the Dependency Graph, Set of states, and rules sets presented above to illustrate how multiples-sites defense analysis can be achieved in anticipation games thanks to strategies. To do so we consider the two following cases. In the first case the company's network does not rely on honey-net information to detect zero day attacks and therefore the honey-net is removed from the simulation. In the second case, the interaction between the honey-net and the company's network occurs. This is the exact configuration described earlier during the paper. For both cases, we run the analysis to find the following administrator strategy objectives:

$$S : (\text{Defense strategy}, Admin, MIN(Cost) \wedge MAX(OCost), OCost > Cost, \Box\neg Compr, \neg 2)$$

This is the administrator dominant strategy objective as introduced in 4. Accordingly we analyze the following attacker strategy objectives:

$$S : (\text{Attack strategy}, Intruder, MAX(Reward) \wedge MAX(OCost), Reward > OCost, \dagger Compr)$$

Theses objectives are used to find the strategy that ends-up with the maximum of hosts compromised even under the best opponent strategy (strict dominance) or at least against the maximum number of opponent strategies (weak dominance).

7.1 Single Site Defense

When the honey-net is not present the only type of attack that can defeated is the public exploit attack by patching the vulnerable service as soon as the patch is available:

Ts	Pl	Ac	Rule	Ta	S	Pa	C
0	I	sel	0day avail	2	⊥	-	-
48	I	**exec**	0day avail	2	⊥	0	0
48	I	sel	Custom avail	2	⊥	-	-
336	I	**exec**	Custom avail	2	⊥	0	0
337	I	sel	Public avail	2	⊥	-	-
337	A	sel	Patch avail	2	⊥	-	-
385	I	**exec**	Public avail	2	⊥	0	0
385	I	sel	Compr public	7	2	-	-
385	A	**exec**	Patch avail	2	⊥	0	2700
385	A	sel	Patch	7	2	-	-
391	A	**exec**	Patch	7	2	1	3500
392	I	`fail`	Compr public	7	2	0	200

Rule names have been abbreviated. Column abbreviations are Ts for time, Pl for player, Ac action, Ta target vertex, S successor vertex, Pa payoff and C for cost.

This weakly dominant strategy dominates only public exploit attack. Custom exploit and zero day attack are unstoppable. The situation can be improved by adding the following rule:

$$\Gamma_{3:1}: \quad \textbf{Pre}\ \Diamond_{\gg} CustomAvail \wedge Vuln \wedge Public \longrightarrow 0, \text{A, Firewall, 100} \quad \textbf{Effect}\ Public$$

This rule models that the administrator can decide to take preventive actions and firewall vulnerable services as soon as a vulnerability is disclosed. With this rule the administrator weakly dominant strategy improves as it now successfully defeats custom exploit:

Ts	Pl	Ac	Rule	Ta	S	Pa	C
0	I	sel	0day avail	2	⊥	-	-
48	I	**exec**	0day avail	2	⊥	0	0
48	I	sel	Custom avail	2	⊥	-	-
336	I	**exec**	Custom avail	2	⊥	0	0
337	I	sel	Public avail	2	⊥	-	-
337	A	sel	Patch avail	2	⊥	-	-
385	I	**exec**	Public avail	2	⊥	0	0
385	I	sel	Compr custom	7	2	-	-
e 385	A	**exec**	Patch avail	2	⊥	0	3600
385	A	sel	Firewall	7	2	-	-
388	A	**exec**	Firewall	7	2	0	5600
388	A	sel	Patch	7	2	-	-
389	I	`fail`	Compr custom	7	2	0	2000
394	A	**exec**	Patch	7	2	1	6100
394	A	sel	UnFirewall	7	⊥	-	-
395	A	**exec**	UnFirewall	7	⊥	1	6403

The intruder still has a dominant strategy that involves using 0day exploit. Additionally firewalling service as soon as a vulnerability is disclosed is in most cases not a suitable policy.

7.2 Multiple-sites Defense

When the honey-net is used the administrator defense strategy can defeat zero day attacks as long as the honey-net is targeted first:

Ts	Pl	Ac	Rule	Ta	S	Pa	C
0	I	sel	0day avail	2	$\perp$	-	-
48	I	**exec**	0day avail	2	$\perp$	0	0
48	I	sel	Compr 0 day	4	2	-	-
51	I	**exec**	Compr 0 day	4	2	1	20000
52	I	sel	Compr 0 day	7	2	-	-
52	A	sel	Attack catched	4	$\perp$	-	-
52	A	**exec**	Attack catched	4	$\perp$	0	2000
52	A	sel	Firewall	7	4	-	-
52	A	**exec**	Firewall	7	4	0	4800
54	I	`fail`	Compr 0 day	7	2	1	40000
54	I	sel	Custom avail	2	$\perp$	-	-
342	I	**exec**	Custom avail	2	$\perp$	1	40000
343	I	sel	Public avail	2	$\perp$	-	-
343	A	sel	Patch avail	2	$\perp$	-	-
390	I	**exec**	Public avail	2	$\perp$	1	40000
391	A	**exec**	Patch avail	2	$\perp$	0	4000
391	A	sel	Patch	7	2	-	-
397	A	**exec**	Patch	7	2	1	4500
397	A	sel	UnFirewall	7	$\perp$	-	-
398	A	**exec**	UnFirewall	7	$\perp$	1	4803

But the intruder has still a strictly dominant strategy that involves to attack company's network first:

Ts	Pl	Ac	Rule	Ta	S	Pa	C
0	I	sel	0day avail	2	$\perp$	-	-
48	I	**exec**	0day avail	2	$\perp$	0	0
48	I	sel	Custom avail	2	$\perp$	-	-
336	I	**exec**	Custom avail	2	$\perp$	0	0
337	I	sel	Public avail	2	$\perp$	-	-
337	A	sel	Patch avail	2	$\perp$	-	-
385	I	**exec**	Public avail	2	$\perp$	0	0
385	I	sel	Compr 0 day	4	2	-	-
385	A	**exec**	Patch avail	2	$\perp$	0	5700
385	I	sel	Compr custom	7	2	-	-
385	A	sel	Patch	7	2	-	-
389	I	**exec**	Compr custom	7	2	1	2000
389	I	sel	Trust abuse	8	7	-	-
391	I	**exec**	Trust abuse	8	7	2	2200
391	I	sel	Trust abuse	6	7	-	-
391	A	**exec**	Patch	7	2	1	10400
393	I	**exec**	Trust abuse	6	7	3	2400
393	I	sel	Compr public	4	2	-	-
400	I	**exec**	Compr public	4	2	4	2600
401	A	sel	Attack catched	4	$\perp$	-	-
401	A	**exec**	Attack catched	4	$\perp$	1	14200

This result is consistent with real world honey-net purpose that aims at reducing the threat by catching unknown threats without being able to catch them all. Note that the intruder strategy uses a custom exploit and not the zero day exploit because it is sufficient to compromise the network if the intruder targets first the company's network.

7.3 Further zero day threat mitigation discussion

Using a honey-net allows to defeat partially zero day attacks. It is possible to extend this model to n sites to mitigate even more zero day threats. In order to compute the risk associated to an zero day attack, it is assumed that zero day attacks are only used against very specific sites, and that the attacker cannot make a distinction between the real and the honey-net service. Under these hypothesis, with one honey service and one real service, the probability that the attacker targets first the honey service is $\frac{1}{2}$. Therefore adding a honey service allows to mitigate the zero day threat by 50%. Adding an extra honey-net service will reduce the zero day threat by 66% and so on. We call this **absolute threat reduction**. Another possibility is to use multiple-sites cooperation to achieve **relative threat reduction**. Relative threat reduction does not focus on reducing the threat that all services experience but the threat that each service experiments. If we add to the example an other site which has the same real service and the same honey-net service then the absolute threat remains the same because the probability that the attacker targets first the honey-net is still 50% (2 of 4). However if we add an interaction between sites that allows to alert other sites when a real service is compromised then the relative threat that each service experience is reduced to 25%. Relative threat reduction is used to improve network resilience to attack. This directly applies to sites that use multiples mirrors. It can also apply to P2P networks where one node can alert others to mitigate the attack propagation.

8 Evaluation

To evaluate the effectiveness of anticipation games to analyze complex multiple-sites scenarios, we have implemented the full framework with locations and penalties in a tool called NetQi [6]. The game engine is written in C for performance reasons. Evaluations were conducted on a regular Linux core 2 desktop using the standard *time* command and the game engine built-in stats output to know how many plays and states have been considered.

In order to measure the scalability of the approach, we have taken the example presented in the paper as a basis and extended it by adding more sites. The evaluation scenario considered is when these sites rely on a single honey-net to mitigate zero day threat. This type of scenario occurs for instance when multiples companies outsource their network security to the same company. This scenario analysis is

possible because the set of rules allows a simultaneous use of the firewall rule. The example extension was done by increasing the number of services by site to 10, and duplicating the company site until the number of sites considered is sufficient. When a site is added, we also add its corresponding set of dependencies to the virtual vertex and the honey-net to the model. For instance a 3 sites analysis involves a honey-net and two company's networks (30 services). We have also evaluated the impact of increasing the number of vulnerability by site. Finally we have studied the impact of rule interleaving by running the analysis with one attack rule (0 day) then two rules (zero day and custom), and finally with the three rules of the example. We do not detail the impact of incrementing the number services by site here because it have a lesser impact on performance than the others criterion. The analyzer was asked to find the defense strategy.

$Sites$	$Rules$	$Time$
2	1	0.03
3	1	0.08
4	1	8.24
5	1	16517
2	2	0.03
3	2	0.08
4	2	12.23
5	2	23970
2	3	0.03
3	3	0.1
4	3	1020

Figure 7. Site and rules number impact on performance

Even if the logic is EXPTIME-hard it is possible to perform an analysis on five sites (50 services) for one vulnerability in a reasonable amount time as presented in figure 7. The time needed to performs the analysis explode when 6 sites are considered. We were able to contains time blow until 5 sites because NetQi take advantage of strategy information to perform early cuts. The idea behind early cuts is to stop considering a play as soon as the strategy constraint is violated. This optimization only applies when the strategy has a path constraint ($\Box$). In our defense strategy it works because as soon as a service that does not belong to the honey-net is compromised, the execution is cut. This optimization is very effective because it scales with the number of sites. The impact of the optimization is visible in figure 8 that presents analysis performance results for 3 sites, 1 vulnerability, and 1 attack rules. The number of states and plays considered greatly decreases.

Early cut	plays	states	time (s)
No	6 113 459	18 444 859	515
Yes	124 047	366 829	9

Figure 8. Early cut impact evaluation

The impact of increasing the number of vulnerabilities with two sites (paper exact setup) is depicted in figure 9. As soon as the number of vulnerabilities grow the time complexity begin to blow.

Nb of vuln	time (s)
1	0.03
2	2
3	3316
4	>week

Figure 9. Vulnerability number impact on performance

9 Conclusion

We have introduced an extension for anticipation games that adds locations and penalties. This extension can be used to model multiple-sites defense scenario. We also have proved that this extension does not change anticipation games complexity. The performance evaluation of anticipation games with this extension done with our tool NetQI shown that even if anticipation games are EXPTIME-complete they can still be used in practice to model complex scenario. As a future direction of work, we will try to find a suitable service collapsing abstraction, to contain even more time explosion.

References

[1] B. Adler, L. de Alfaro, and M. Faella. Average reward timed games. In *FORMATS 05*, volume 3829, pages 65–80. Springer-Verlag, 2005.

[2] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

[3] M. Artz. *NetSPA : a Network Security Planning Architecture*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science., 2002.

[4] M. R. B. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.

[5] J. Balthrop, S. Forrest, M. E. J. Newman, and M. M. Williamson. Technological networks and the spread of computer viruses. *Science*, 304:527–529, Apr 2004.

[6] E. Bursztein. Netqi http://www.netqi.org.

[7] E. Bursztein and J. Goubault-Larrecq. A logical framework for evaluating network resilience against faults and attacks. In *12th annual Asian Computing Science Conference (ASIAN)*, pages 212–227. Springer-Verlag, Dec. 2007.

[8] E. Bursztein and J. C. Mitchell. Using strategy objectives for network security analysis. In *5th International Conferences on Information Security and Cryptology INSCRYPT*. Springer-Verlag, 2009.

[9] A. Church. Logic, arithmetics and automata. In *Congress of Mathematician*, pages 23–35. Institut Mittag-Leffler, 1962.

[10] V. Colizza, A.Barrat, M. Barthelemy, and A. Vespignani. The modeling of global epidemics: stochastic dynamics and predictability. *Bulletin of Mathematical Biology*, 68:1893–1921, 2006.

[11] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *Symposium on Research in Security and Privacy*, pages 202–216. IEEE Computer Society, 2002.

[12] F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 197–216, London, UK, 2000. Springer-Verlag.

[13] M. Dacier, Y. Deswarte, and M. Kaaniche. Models and tools for quantitative assessment of operational security. In *12th International Information Security Conference*, pages 177–186, May 1996.

[14] L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *14th International Conference on Concurrency Theory*, volume 2761 of *LNCS*, pages 144–158. Springer-Verlag, 2003.

[15] T. Henzinger and V. Prabhu. Timed alternating-time temporal logic. In *Formats 06*, volume 4202, pages 1–18. Springer-Verlag, 2006.

[16] K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 121–130, Washington, DC, USA, 2006. IEEE Computer Society.

[17] S. Jha, O. Sheyner, and J. Wing. Two formal analysis of attack graphs. In *CSFW '02: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 49–63, Washington, DC, USA, 2002. IEEE Computer Society.

[18] F. Laroussinie, N. Markey, and G. Oreiby. Model-checking timed atl for durationnal concurrent game structures. In *FORMATS 06*, volume 4202 of *LNCS*, pages 245–259. Springer-Verlag, 2006.

[19] R. Lippmann, S. Webster, and D. Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *RAID '02: Proceedings of the 5th International Workshop on Recent Advances in Intrusion Detection*, pages 307–326. Springer-Verlag, Oct 2002.

[20] K.-w. Lye and J. M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005.

[21] A. Mahimkar and V. Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *18th IEEE Computer Security Foundations Workshop (CSFW), Aix-en-Provence, France, June 2005, pp. 287-301. IEEE Computer Society, 2005.*, pages 287–301. IEEE Computer Society, Jun 2005.

[22] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (extended abstract). In *STACS 95*, pages 229–242, 1995.

[23] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2d2 : a formal data model for ids alert correlation". In *RAID Recent Advances in Intrusion Detection*, LNCS, pages 115–127. Springer-Verlag, 2002.

[24] S. Noel and S. Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In

VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, pages 109–118, New York, NY, USA, 2004. ACM Press.

[25] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference*, pages 86–95, Dec. 2003.

[26] J. Pamula, S. Jajodia, P. Ammann, and V. Swarup. A weakest-adversary security metric for network configuration security analysis. In *QoP '06: Proceedings of the 2nd ACM workshop on Quality of protection*, pages 31–38, New York, NY, USA, 2006. ACM Press.

[27] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190, New York, NY, USA, 1989. ACM Press.

[28] C. Ramakrishan and R. Sekar. Model-based analysis of configuration vulnerabilities. In *Journal of Computer Security*, volume 1, pages 198–209, 2002.

[29] E. Rasmusen. *Games and Information*. Blackwell publishing, 2007.

[30] R. W. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 156–165, Washington, DC, USA, 2000. IEEE Computer Society.

[31] F. Saffre, J. Halloy, and J. L. Deneubourg. The ecology of the grid. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 378–379, Washington, DC, USA, 2005. IEEE Computer Society.

[32] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's journal*, Dec. 1999.

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

[34] R. Segala, R. Gawlick, J. F. Sogaard-Andersen, and N. A. Lynch. Liveness in timed and untimed systems. *Inf. Comput.*, 141(2):119–171, 1998.

[35] H. R. Shahriari and R. Jalili. Modeling and analyzing network vulnerabilities via a logic-based approach.

[36] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated generation and analysis of attack graphs. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273 – 284, Washington, DC, USA, 2002. IEEE Computer Society.

[37] L. P. Swiler. A graph-based network-vulnerability analysis system. In *New Security Paradigms Workshop*, pages 71 – 79. ACM Press, 1998.

[38] S. J. Templeton and K. Levitt. A requires/provides model for computer attacks. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, New York, NY, USA, 2000. ACM Press.

[39] V.Gorodetski and I.Kotenko. Attacks against computer network: Formal grammar-based framework and simulation tool. In *5th International Conference "Recent Advances in Intrusion Detection"*, pages 219–238. Springer-Verlag, 2002.

[40] M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *18th Annual Computer Security Applications Conference*, pages 61–68, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[41] D. Zerkle and K. Levitt. Netkuang: a multi-host configuration vulnerability checker. In *SSYM'96: Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography*, pages 195–201. Usenix, 1996.