Extending Anticipation Games with Location, Penalty and Timeline

Elie Bursztein (eb@lsv.ens-cachan.fr)

LSV, ENS Cachan, CNRS, INRIA, France

Abstract. Over the last few years, attack graphs have became a well recognized tool to analyze and model complex network attack. The most advanced evolution of attack graphs, called anticipation games, is based on game theory. However even if anticipation games allow to model time, collateral effects and player interactions with the network, there is still key aspects of the network security that cannot be modeled in this framework. Theses aspects are network cooperation to fight unknown attack, the cost of attack based on its duration and the introduction of new attack over the time. In this paper we address these needs, by introducing a three-fold extension to anticipation games. We prove that this extension does not change the complexity of the framework. We illustrate the usefulness of this extension by presenting how it can be used to find a defense strategy against 0 days that use an honey net. Finally, we have implemented this extension into a prototype, to show that it can be used to analyze large networks security.

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, network cooperation 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 [5] is the most advanced evolution of attach graphs, based on game theory [7] and more specifically on TATL [8] (Timed Alternating-time Temporal Logic).

Modeling network cooperation strategy cannot be achieved 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 network communication. For example, patching rules should not be applied to a 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.A timeline of events is a succession of events that need to take place one after the other.

We also extend anticipation games with penalties. Intuitively a penalty is a cost added for each unit of time a constraint holds. As additional benefit penalty allow to model another 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. 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.

The main contribution of this paper is an extension of the anticipation game framework that allows to model network cooperation, intrusion cost based on their duration, and the introduction of new event over the time. To the best of our knowledge, our extended framework is the first which is able to model and analyze those aspect of network security. We prove that anticipation games with locations and penalties are decidable and that the complexity of the model remains EXPTIME-complete. We also have implemented our framework into an freely available tool called NetQi [3] to evaluate the effectiveness of the approach. In the evaluation section we will show that it is possible to analyze complex multiple-sites scenarios even on large networks with thousand of services. To illustrate how our extension works, we provide a running example that discusses a network multiple-sites defense strategy that uses of an honey-net against various types of exploits including zero day ones.

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 timeline of events is illustrated. Sect. 6 introduces the notion of penalty and cost diminishing. Sect 7 covers the multiple-sites defense strategies that were found by analyzing the running example with our prototype. In sect. 8 we evaluate the effectiveness of the approach.We conclude in Sect. 9

2 Related Work

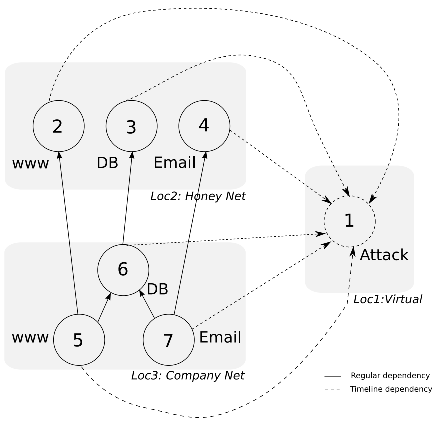
Model checking for attack graphs was introduced by Ammann and Ritchey [17]. They are used to harden security [14]. Various methods have been proposed for finding attack paths, i.e., sequences of exploit state transitions, including logic-based approaches [15, 19, 9, 18], and graph-based approaches [21, 20, 13]. Anticipation game are based on timed automata, timed games, and timed alternating-time temporal logic (TATL) [8], a timed extension to alternating-time Kripke structures and temporal logic (ATL) [1]. The TATL framework was specifically introduced in [7]. Game strategies have been used to predict players actions in numerous domains ranging from economy to war [2, 16]. the notion of cost diminishing appears in [6]. The use of games for network security was introduced by Lye and Wing [11]. The anticipation game framework was presented by Bursztein and Goubault-Larrecq [5] and network strategies for anticipation games were detailed in [4]. Finally the first use of game theory for denial of service was done by Mahimkar and Shmatikov [12].

3 Anticipation Games

An anticipation game is a timed game [7], the key difference between standard timed games and anticipation games is the dual-layer structure used in anticipation games. Its lower-layer called the **Network Layer** is used to represent network information. Its upper-layer called the **Attack Layer** is a regular TATL game structure used to model the network state evolution induced by players actions such as exploiting a vulnerability. Anticipation games can be thought as a graph of graphs where the lower graph is the network state and the above graph describes the transition between one network state to an other. The players of an anticipation games 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 transition represents the execution of one rule. In an anticipation game a 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. Using a network initial configuration and a set of rules, an anticipation game is used to answer questions such as : *what should I do to counter this type of attack ?* A questions is represented by **strategies objectives** and its answer, the **strategy**, is the play that fulfill best these objectives. Strategies objectives are composed of two main parts: a set of constraints and a set of goals. Constraints are used to express conditions on the network state that a play must satisfy to be considered as a potential strategy. A typical defense strategy constraint is that no service is ever compromised during the play and that at the end of the play no service is vulnerable anymore. Goal are used to select among all the play that satisfy the set of constraints the one that is the most relevant by analyzing the cost, reward and time outcome. A typical defense strategy goal is to minimize the cost of the strategy. The cost reward and timing used in this paper are not meant to be realistic, they are only here for example purpose. While interesting, computing the real value of cost, reward and timing is out of the scope of this paper that aims at providing a mean to reason on them.

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. Formally, let $\mathcal{A}$ be a finite set of so-called *atomic propositions* $A_1, \ldots, A_n, \ldots$, denoting each base property. Each atomic proposition is true or false at each vertex. E.g., Vuln is true at each vertex that is vulnerable. Thus each atomic proposition is true or false for each of Dependency Graph vertices. *States* on Dependency Graph are then simply functions $\rho : \mathcal{A} \rightarrow \mathbb{P}(V)$ mapping each atomic proposition to the set of vertices that satisfies it. We describe ρ in a finite way, as a table of all pairs $(A, v) \in \mathcal{A} \times \mathbb{P}(V)$ such that $v \in \rho(A)$; hence there are finitely many states. The Dependency Graph used as an example and the corresponding set of states are represented in Figure 1.



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)	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Fig. 1. Dependency Graph (left) and Initial set of states (right)

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 for the timeline of events with a dashed line. The role of the virtual vertex and its incoming dependencies is to model the 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)*, which is a web server, depends on the vertex *DB (6)*, which is a database, to retrieve user credential for authentication purpose. 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 fake honey-net services are used to lure the attacker and catch him when he tries to attack them. Using an honey-net allow to catch unknown attack because the only traffic they get are attacks. Hence if the traffic is not a known attack, it is likely that its a new type of attack. That is why each company's services depend on its honey-net fake twin service to defeat a zero day attack.

The complete set of variables mapping used in the example can be divided into three parts. The first part is variables `0DayAvail`, `CustomAvail`, `PubAvail` and `PatchAvail` which are used to model the timeline of events.The second part is variables `Detected` and `Monitored` which are used for multiple-sites defense purpose. Finally the third part is used to describe the network's initial state. The variable `Vuln` is used to model that the company web and email services along with their fake twin services located on the honey-net are vulnerable to an unknown vulnerability, The variable `Compr` is used to say that no service is compromised. Finally the variable `Public` is used to indicate that every service is public (not firewalled).

3.2 Attack Layer

TATL [8] extends ATL [1] with the notion of timed game by adding time cost to transitions. From the network security perspective this is important because it models 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. This time race introduces a so called element of surprise [7]. For example the intruder can take the administrator by surprise if he can exploit a vulnerability faster than the administrator can patch it. This is coherent with real network security where you cannot foresee what attacker will come up next.

3.3 Rules of the game

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

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

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 who 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 [4]. **Granting rule** use the $\Longrightarrow$ double arrow and **regular rules** use 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 rules do not grant any reward. Regular rules are used for temporary actions and for the 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 modal 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 have restored the successor vertex. In this case, the intruder is taken by surprise, and the compromise rule fails.

4 Location

In the original anticipation games [5] a rule can be applied to any Dependency Graph vertex as long as its set of constraints meets the rule preconditions requirements. However in many cases such behavior is not suitable. In particular it is not possible to model network multiple-sites defense analysis without restricting the scope of rules. This impossibility is mainly due to the fact that different rules need to be applied to the different networks. Therefore we distinguish three type of rules: the **transitional** ones, the **local** ones, and the **global** ones. Transitional 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 timeline of events rules to the virtual location. Finally global rules are meant to be used on any vertex. Similarly strategies objectives might 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. A location is a non-empty set 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 2 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 (Transitional rule) *A rule is transitional 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

1) $\mathit{\Gamma}_{:}$: $\textbf{Pre}$: $\Diamond 0DayAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 3, I, 0 day exploit, 20000
$\textbf{Effect}$: $Compr$

2) $\mathit{\Gamma}_{:}$: $\textbf{Pre}$: $\Diamond CustomAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 4, I, Custom exploit, 2000
$\textbf{Effect}$: $Compr$

3) $\mathit{\Gamma}_{:}$: $\textbf{Pre}$: $\Diamond PubAvail \wedge Vuln \wedge Public \wedge \neg Compr$
$\Longrightarrow$ 7, I, Public exploit, 200
$\textbf{Effect}$: $Compr$

4) $\mathit{\Gamma}_{3:3}$: $\textbf{Pre}$: $\neg Compr \wedge \Diamond Compr$
$\Longrightarrow$ 2, I, Trust Abuse, 200
$\textbf{Effect}$: $Compr$

5) $\mathit{\Gamma}_{1:1}$: $\textbf{Pre}$ $Monitored \wedge Compr \wedge \neg Detected$
$\longrightarrow$ 1, A, Attack Detected, 2000
$\textbf{Effect}$ $Detected$

6) $\mathit{\Gamma}_{2:2}$: $\textbf{Pre}$ $\neg Vuln \wedge \neg Public$
$\longrightarrow$ 1, A, Unfirewall, 100
$\textbf{Effect}$ $Public$

7) $\mathit{\Gamma}_{3:2}$: $\textbf{Pre}$ $\Diamond Detected \wedge Vuln \wedge Public$
$\longrightarrow$ 0, A, Firewall, 100
$\textbf{Effect}$ $\neg Public$

8) $\mathit{\Gamma}_{3:1}$: $\textbf{Pre}$ $\Diamond PatchAvail \wedge Vuln$
$\longrightarrow$ 6, A, Patch, 500
$\textbf{Effect}$ $\neg Vuln$

Fig. 2. Set of rules used to model a players action

preconditions ensure that the targeted 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 ($\mathit{\Gamma}$ has no index). They differ because due to the events timeline, they are available at a different time. 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 constraint 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 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 us to model the trade-off between the advantage awarded by an undisclosed vulnerability exploit and the investment required to find it.

4.3 Local Rules

Rules 4, 5, and 6 are local rules. Their $\mathit{\Gamma}$ index is of the form $n : n$ where the first n is the vertex location and the second n is the successor location. 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 diagram 1 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. This 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. 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 Transitional Rules

Rules 7 and 8 are transitional 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 transitional 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 timeline of events evolves. The timeline information is needed to know when the patch is available. This rule can only be transitional: if it is global, it can be applied to honey-net and if it is local it does not work because the timeline of events evolution take place in the virtual location.

4.5 Strategy with location

Definition 4 *(***Strategy***) A strategy is the tuple* $S : (\texttt{name}, \mathsf{P}, \mathcal{O}, \mathcal{R}, \mathcal{C}, \mathcal{L})$ *where* `name` *is the strategy name,* P *its owner,* $\mathcal{O}$ *is the strategy objectives set,* $\mathcal{R}$ *is the objectives priority strict order,* $\mathcal{C}$ *is the set of constraints for the play and* $\mathcal{L}$ *is the set of constraint for the location.*

In our example the following defense strategy objectives are used:

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

They are used to find the play for the administrator that primarily maximizes intruder cost ($MIN(OCost)$), and secondarily minimizes the administrator cost ($MIN(Cost)$), and ensures that no service in every location except the honey-net location ($\neg 2$) is ever compromised ($\Box\neg Compr$). Adding the opponent cost maximization objective aims at finding the **(weakly) dominant strategy**.

Definition 5 (weakly) dominant strategy *A dominant strategy is the strategy that beats every opponent strategy (strict dominance) or at least maximizes the number of strategies beaten (weak dominance).*

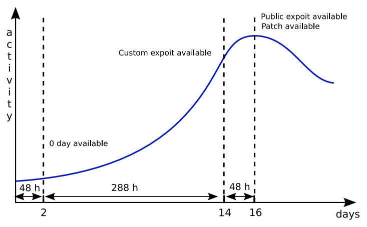
The strategy returned for these objectives can be view as the play where the opponent plays his best game against the targeted player. Model-checking strategies constraints against Anticipation games with location is still decidable (see proof in appendix **??**)

Lemma 1. *Model-checking strategies constraints against Anticipation games extended with locations is decidable.*

5 Using a Timeline of Events

Being able to model a timeline of events is mandatory because many network security scenarios need it. For instance the classical vulnerability cycle [10] follows a 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 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 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 timeline of events state in rule preconditions and effects (as in the Dependency Graph depicted in figure 1). Locations are used to ensure that the virtual vertex is the only one used in 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



1) $\Gamma_{1:1}$: $\textbf{Pre}\ \neg 0DayAvail$
$\longrightarrow$ 48, I, O day exploit Available, 0
$\textbf{Effect}\ 0dayAvail$
2) $\Gamma_{1:1}$: $\textbf{Pre}\ \neg CustomAvail \wedge 0DayAvail$
$\longrightarrow$ 288, I, Custom exploit available, 0
$\textbf{Effect}\ CustomAvail$
3) $\Gamma_{1:1}$: $\textbf{Pre}\ \neg PubAvail \wedge CustomAvail$
$\longrightarrow$ 48, I, Public exploit available, 0
$\textbf{Effect}\ Pub$
4) $\Gamma_{1:1}$: $\textbf{Pre}\ \neg PatchAvail \wedge CustomAvail$
$\longrightarrow$ 48, I, Patch available, 0
$\textbf{Effect}\ 0dayAvail$

Fig. 3. Vulnerability timeline of events (left) and the set of rules used to model timeline evolution (right)

The multiple-site defense example uses a timeline of events inspired by the standard vulnerability cycle represented in diagram 3 which is modeled by the four sets and four rules presented in figure 3. Intuitively in this model 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 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.

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:

$$\varGamma_{1:2} : \mathbf{Pre}\ Detected \xrightarrow{\text{288, A, Reverse Engineering , 500}} \mathbf{Effect} CustomAvail$$

This transitional 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.

Another type of 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 2. 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 [4], 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 has a cost. However this approach has an important limitation: it does not allow to model costs that are time dependent. Such cost exists for on-going processes which are prominent in network security. Two well known examples of such on-going processes are service DOSing (Denial Of Service)[12], and intrusion detection monitoring. The longer they last, the higher the cost is. To model this type of cost, anticipation games need to be extended with the notion of 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 6 *(***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 $\mathsf{F(x)}$ is the function $\mathsf{F(x)} : \mathbb{N}^* \rightarrow \mathbb{N}$ that takes as parameter the integer x which is the number of units of time elapsed since the penalty has been triggered and returns the corresponding cost for this unit of time. The total cost generated by the penalty is therefore the sum of all the costs returned by the penalty function.*

Here is 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 (¬ *Avail*) because of the DOS, the company loses \$1000 of income. This can be modeled by adding the following penalty to the game:

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

which states that for each unit of time where the vertex 5 (www service) is not available the administrator cost is increased by 1000. The use of such penalty allows the incident strategy cost minimization objective to take into account the relation between the loss of income and the time elapsed. In the running 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 to use various cost models such as an exponential cost or a diminishing cost model as presented above.

Another important time/cost relation to consider is when the cost diminishes over the time [6]. 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 the action of 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 timeline of events to ensure that the cheaper rule is only used after the most expensive one has been used. To model a diminishing supervision cost with a lower bound the following kind of penalty can be used:

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

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 kinds 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 three penalties to the analysis, one for each honey-net service. For example the following penalty is added for the vertex 2:

$$P : (\text{Administrator}, 2, \text{Monitored}, f(x) \rightarrow 10)$$

6.1 Decidability and Complexity of the extended model

Even if adding penalty allows to model a brand new range of cost, from the decidability perspective, extending the framework with penalty does not change the decidability (see proof in appendix **??**)

Lemma 2. *Model-checking strategies constraints against Anticipation games extended with penalty is decidable.*

From Lemma 1 and Lemma 2 it follows:

Theorem 1 *Model-checking strategies constraints against Anticipation games extended with penalties and locations is decidable*

Which is the central theoretical result. Additionally we prove that locations and penalties does not change the anticipation games complexity bound (see proof in appendix **??**) which is a key result for the practicality of the approach:

Theorem 2 *Model-checking strategies constraints over anticipation games extended with penalties and locations remain EXPTIME-Complete*

7 Multiple sites Strategies Illustration

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 administrator dominant strategy objective as introduced in 4:

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

When the honey-net is not present the only type of attack that can be countered is the public exploit attack one. This is done by patching the vulnerable service as soon as the patch is available. This defense strategy is presented in figure 4. In this figure, 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. `A` denote the administrator player and `I` for the intruder.

The row one of the table on the left is read as follows: at time `0` the Intruder (`I`) selects (`sel`) the rule `0day avail` on vertex `2`, there is no successor involved ($\perp$). The intruder reward and cost are not yet intialized (`-`). Accordingly the line 2 states that at time `48` the intruder (`I`) execute (`exec`) the rule `0day avail` on vertex 2, his current cost is `0` and his current reward is `0`. And so on.

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

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	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	⊥	-	-
52	A	**exec**	Attack catched	4	⊥	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	⊥	-	-
342	I	**exec**	Custom avail	2	⊥	1	40000
343	I	sel	Public avail	2	⊥	-	-
343	A	sel	Patch avail	2	⊥	-	-
390	I	**exec**	Public avail	2	⊥	1	40000
391	A	**exec**	Patch avail	2	⊥	0	4000
391	A	sel	Patch	7	2	-	-
397	A	**exec**	Patch	7	2	1	4500
397	A	sel	UnFirewall	7	⊥	-	-
398	A	**exec**	UnFirewall	7	⊥	1	4803

Fig. 4. Defense strategy without Honey-net (left) Defense strategy with honey-net (right)

The defense efficiency can be improved by taking preventive action when the vulnerability is disclosed (See appendix **??**). When the honey-net is used, the defense strategy can mitigate zero day attacks as long as the honey-net is targeted first by the intruder, as detailed in figure 4 (on the right). Even with the introduction of an honey-net, the intruder has a strictly dominant strategy that involves attacking company's network services first (see appendix **??**). This is consistent with real world honey-net purpose that aims at mitigate 0 day attack by catching unknown threats without the guaranty catch them all.

8 Evaluation

To evaluate the effectiveness of anticipation games to analyze complex multiple-sites scenarios, we have implemented the full framework in a tool written in C for performance reasons. Evaluations were conducted on a Linux core 2 desktop using the tool built-in benchmark option. The game used in the evaluation is the one presented in this paper with more company networks and more services per network. Benchmark results are summarized in the table below. Time is in second. The prototype includes many optimizations to delay the execution time blowup. It follows that it is possible to find the optimal strategy for 50 services divided into 4 sites and 1 honey-net. When more services are added, the execution time blowup as predicted by the theoretical complexity bound. That is why for larger network, we have designed an heuristic that is able to find an approximate strategy by using a dynamic rules ordering algorithm. The strategy returned by this algorithm is sound, it satisfies the strategy constraint, but there is no guarantee that it is the best one. However on small examples, it appears to be so. This evaluation shows that anticipation games is suitable to analyze complex scenarios even on very large networks.

Analysis	Num of service	Num of network	Analysis time in sec
Exact	30	2	0.03
Exact	40	3	0.1
Exact	50	4	1020
Appro	2000	1	0.48
Appro	5000	4	0.82
Appro	10000	3	2.26

9 Conclusion

We have introduced an extension for anticipation games that allows to analyze network cooperation and cost over the time. We have also proved that this extension does not change anticipation games complexity. Finally we have shown with our prototype that anticipation games with this extension can be used in practice to model complex scenario even when each network have thousand services. As a future direction of work, we will focus on dependency graph static analysis to improve the scalability of the exact solution.

Acknowledgements

Thanks to Richard Lippmann for his advices on multi-sites defense and Marc Dacier for the discussion about the notion of cost diminishing.

References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
2. M. R. B. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.
3. E. Bursztein. Netqi http://www.netqi.org.
4. E. Bursztein. Network administrator and intruder strategies. Technical Report LSV-08-02, LSV, ENS Cachan, Jan 2008.
5. 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.
6. 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.
7. 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.

8. T. Henzinger and V. Prabhu. Timed alternating-time temporal logic. In *Formats 06*, volume 4202, pages 1–18. Springer-Verlag, 2006.
9. 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.
10. 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.
11. K.-w. Lye and J. M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005.
12. 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.
13. 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.
14. 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.
15. C. Ramakrishan and R. Sekar. Model-based analysis of configuration vulnerabilities. In *Journal of Computer Security*, volume 1, pages 198–209, 2002.
16. E. Rasmusen. *Games and Information*. Blackwell publishing, 2007.
17. 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.
18. H. R. Shahriari and R. Jalili. Modeling and analyzing network vulnerabilities via a logic-based approach.
19. 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.
20. L. P. Swiler. A graph-based network-vulnerability analysis system. In *New Security Paradigms Workshop*, pages 71 – 79. ACM Press, 1998.
21. 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.