Using Strategy Objectives for Network Security Analysis

Elie Bursztein[1] and John C. Mitchell[2]
{elie|mitchell}@cs.stanford.edu

[1] Stanford University and LSV, ENS Cachan
[2] Stanford University

Abstract. The anticipation game framework is an extension of attack graphs based on game theory. It is used to anticipate and analyze intruder and administrator concurrent interactions with the network. Like attack-graph-based model checking, the goal of an anticipation game is to prove that a safety property holds. However using this kind of goal is tedious and error prone on large networks because it assumes that the analyst has prior and complete knowledge of critical network services.

In this paper we address this issue by introducing a new kind of goal called "strategy objectives". Strategy objectives mixes logical constraints and numerical ones. In order to achieve these strategy objectives, we have extended the anticipation games framework with cost and reward. Additionally this extension allows us to take into account the financial dimension of attacks during the analysis. We prove that finding the optimal strategy is decidable and only requires linear space. Finally we show that anticipation games with strategy objectives can be used in practice even on large networks by evaluating the performance of our prototype.

1 Introduction

With the increasing size and complexity of networks, attack modeling is now recognized as a key part of constructing an accurate network security for intrusion analysis, prevention, and forensic. Anticipation games (AG) [7] are an evolution of attack graphs based on game theory. More specifically, an anticipation game is a simultaneous game played between a network attacker and a network defender on a game-board consisting of a dependency graph. The dependency graph defines which services exist on the network and how they are related. The moves of the game do not change this dependency graph, but they do change the attributes, such as the compromise attribute which is associated with the nodes to reflect players action.

Typically an anticipation game is used to analyze how the network will be impacted by various attacks and how administrator actions can counter them. Using Anticipation games instead of attack graphs offers the following advantages:

First it allows us to model the concurrent interaction of the intruder and the administrator with the network. For example it is possible to model a case wheres that the intruder is trying to exploit a vulnerability while the administrator is trying to patch it.

Secondly, player interactions with the network are described by timed rules that use precondition and postcondition written in a modal logic. Describing the model only with the network initial state and a set of rules relieves the administrator from the tedious and error prone burden of explicitly describing each network states and the transitions between them. In AG the model-checking algorithm uses the set of rules to infer automatically every transition and network state reachable from the network initial state. As a result it is possible to express very large and complex models in a very compact form which is handy while working on large networks and complex attacks. Additionally modeling player action by rules allows us to capture security expert reasoning in an intuitive manner.

Thirdly the use of timed rules allows us to model the temporal dimension of the attack. It captures the fact that each interaction with the network requires a different time. For instance, developing and launching an exploit is somewhat slower than downloading and launching a publicly available one. Modeling the time also models the so called "*element of surprise*" [12], which occurs when one player takes the other by surprise because he is faster. For example, when the administrator is patching a service she can be taken by surprise by the intruder if the intruder is able to exploit the vulnerability before the patch is complete.
Finally since AG have been designed for network security analysis, they takes into account network topological information such as dependency between network services, which allow to model *collateral effects*. For example that when a DNS server is unavailable by collateral effect, the web server is merely available because the DNS resolution failed.

1.1 Motivation

Although using AG to analyze attacks provides a substantial improvement over standard attack graphs, there is still one side of attack analysis that remains tedious and error-prone: how to define the analysis goal As in standard attack graph based on model checking [32], the current AG analysis goal is to prove that a given safety property holds for a given model. For example one proves that whatever an intruder does, a given service will never be compromised for a given model. However network analysis, especially when working on large networks, makes the expression of security goal in term of reachability very hard because it is difficult to assert which services/hosts should be considered as primary security objective. Furthermore, this type of goal can't address natural questions that arise while dealing with network security. For example "*What is the most devastating attack against my network*", or "*In the worst case how many services will be compromised ?*" are example of such natural questions.

To answer these questions, one need to be able to express quantitative goals which is not possible when the analysis goal is a safety property. Therefore in this paper we introduce a new kind of analysis goal called "**Strategy objectives**". Intuitively the idea is to combine a symbolic objective (logical formula) with numerical ones (time, cost, and reward).

The logical formula is used to select *all the plays* that are valid strategies, and the numerical objectives are used to refine the analysis by selecting among these possible strategies, *the one* that is the most relevant to the player according to his quantitative objectives.
To the best of our knowledge this is the first time that symbolic and numerical objectives are combined to express player goals. Note that being able to select among all the valid candidates the most relevant one is a central issue in network security as the number of possible candidates (*e.g* different attacks) to achieve a given goal is usually very large. The expressiveness offered by strategy objectives allows anticipation games to be used to a brand new range of question that match more closely administrator and security analysts needs. For example using strategy objective it is possible to answer the question: "*What is the most effective patching strategy in term of cost or time ?*".

Moreover the introduction of action cost and reward takes into account the financial dimension of attacks which is a central concern of network attack. Taking into account action cost allow to reason about the cost required to launch an attack, the loss induced by it, and the investment required to prevent it.

1.2 Contribution

Our main contribution is the extension of AG with strategy objectives. This extension allows the analysis to answer network security key questions. Furthermore it captures the financial dimension of the attack.
As far as we know, with this extension the AG framework is the first attack model that covers both the financial and temporal aspect of attacks. Additionally we prove that the model-checking of AG with strategy objectives is decidable, and that deciding if a play is a valid strategy can be done in linear time. We also prove that using strategy objectives instead of a safety property adds only a linear space complexity to the analysis. Finally we demonstrate with our prototype, that in practice this framework can be used to find strategy for large network (Thousands of nodes), and that practical results are consistent with the theoretical bounds.

The reminder of this paper is organized as follow. In Sect. 2, we will survey related work and in Sect. 3 we recall what an anticipation game is and present how we have extended it to take into account cost and reward. We also present the game example that is used as a guideline for the rest of the paper. Sect. 4 details how strategies objectives are expressed and contains the strategy decidability and space complexity proofs. In sect. 5 we evaluate with our prototype the impact of using strategy in term of speed and memory. We show that experiments are consistent with the theory and that strategies can be used in practice.

2 Related Work

Attack graphs are a very active field pioneered by Schneier [34,35] and Kuang and al [42]. Model checking for attack graphs was introduced by Ammann and Ritchey [32]. They are used to harden security [26]. Various methods have been proposed for finding attack paths, i.e., sequences of exploit state transitions, including logic-based approaches [30,38,30,38,16,37], and graph-based approaches [42,39,25]. Researchs has also been conducted on formal languages to describe actions and states in attack graphs [10], Some rely on grammars [40], some have a more practical focus [9], or specialize on IDS alert correlation [23]. Some authors propose techniques that allow attack graphs to scale to large networks [15]. Security metrics [28] have been developed, and supporting tools such as NetSPA [3] or Sheyner's tool [38] now exist.

The SIR model, which is similar to the compromising recovery cycle, is used to study the propagation of epidemics in biology [8]. Biological models for computer security were proposed recently [33]. As in computer virus propagation research [5,41], 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 [29]. 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 [21]. Anticipation game are based on timed automata, timed games, and timed alternating-time temporal logic (TATL) [14], a timed extension to alternating-time Kripke structures and temporal logic (ATL) [2]. The TATL framework was specifically introduced in [12]. 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 [36]. 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 [17]. Game strategies have been used to predict players actions in numerous domains ranging from economy to war [4,31]. The notion of cost for attack appears in [11]. The use of games for network security was introduced by Lye and Wing [19]. The anticipation game framework was introduced in [7]. Finally Mahimkar and Shmatikov used the game theory to model denial of service in [20].

3 Anticipation Games with Cost and Rewards

This section briefly recalls what an anticipation game is and explains the extension made to introduce strategy in the model. Intuitively, an Anticipation Game (AG) can be represented as a graph. Each node of the graph describes the network state at a given moment, *e.g*, each state describes which services are compromised at this moment. The transitions represent the set of actions that both players, the administrator and the intruder, can perform to alter the network state. For example an edge may represent the action of removing a service from the vulnerable set by patching it.

3.1 Network State

The network state is represented by a graph called *Dependency graph* (DG) and a finite set of states. DG are meant to remain fixed over time and describe the relation between services and files. Figure 3.1 presents the DG used as an example in this paper. DG vertices are services and files present on the network and the set of directed edges is used to express the set of dependency between them. In the example, the direct edge that links the vertex *Email server (5)* to the vertex *User database (6)* is used to denote that the *email server* depends on the *user database* to identify its clients.

States	1	2	3	4	5	6
$\rho(\mathsf{Vuln})$	$\perp$	$\perp$	$\perp$	$\top$	$\top$	$\perp$
$\rho(\mathsf{Public})$	$\perp$	$\perp$	$\perp$	$\top$	$\top$	$\perp$
$\rho(\mathsf{Compr})$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
$\rho(\mathsf{NeedPub})$	$\perp$	$\perp$	$\perp$	$\top$	$\top$	$\perp$

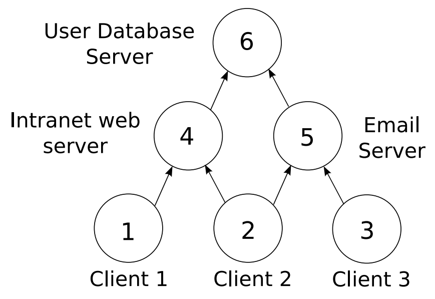

Fig. 1. The intial network state (left) and the dependency graph (right)

The set of *variables* (figure 1) is used to model information that does evolve over time. Intuitively this set describes which services and files are currently public, vulnerable, compromised, and so on. 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 DG vertex. The complete initial mapping used in the example is detailed in figure 1. This mapping indicates that the *email server* and the *web server* need to be public, are vulnerable, and are public because ($\rho(\mathsf{NeedPub})$), $\rho(\mathsf{Vuln})$, and ($\rho(\mathsf{Vuln})$) return true ($\top$) for both of them. It also indicates that no vertex is compromised as $\rho(\mathsf{Compr})$ returns false ($\perp$) for every vertex. Finally the set $\rho(\mathsf{NeedPub})$ is used by the `Unfirewall` rules to know which vertex should be made public.

3.2 Players Actions

To describe which actions are legal for each player a set of timed rules is associated to the AG. Each rule is of the form $\mathbf{Pre}\ F \xrightarrow{\Delta,p,a,c} P$ where F is the *precondition*, stating when the rule applies, Δ is the amount of time needed to fire the rule, p is the name of the player that originates the rule, a is an action name, c is the rule cost, and P is a *command*, stating the effects of the rule. It is required for the precondition F to hold not just when the rule is selected, but during the whole time it takes the rule to actually complete (Δ time units). For example consider the following rule :

$$\mathbf{Pre}\ Vuln \wedge Public \overset{(30,I,Compromise,500)}{\longrightarrow} Compr$$

It says that the intruder can compromise a vertex if it is vulnerable (`Vuln`) and public (`Public`) in 30 units of time. Compromise means here that the targeted vertex will be added to the state `Compr`. If the intruder chooses to apply it to the *Email server* then it is required that the preconditions are fulfilled when he chooses to apply the rule but also after the 30 units of time required to execute it because the network state might have changed due to administrator action. For example the administrator can firewall the targeted vertex. In this case, the vertex is not public anymore, the intruder is taken by surprise, and the compromise rule fails. An AG play is a path (a sequence of action and states) $\rho : s_0 r_0 s_1 r_1 ...$ where $\forall j : s_j \rightarrow^{r_j} s_{j+1}$, s_j and $s_(j+1)$ are network states, and r_j is the rule used to make the transition.

3.3 Extending Anticipation Game for Strategy

Using strategy and analyzing the financial dimension of the attack requires, that we extend the framework. The natural way is to add an action cost to rules and an action reward to each DG vertex:

Cost are added to rules because it is obvious that some action are more costly than others. For example coding an exploit is more costly than using an existing one. Similarly, Rewards are bound to DG vertices because some services and files are more valuable than others. In our example (figure 3.1), it is obvious that the *user database* is more important than any client. Formally we have a function $\mathsf{Value}(x) \rightarrow y / y \in \mathbb{N}$ that returns the value y associated to the DG vertex x. Costs are naturally added to rules because a rule execution is equivalent to a player action on the network. To take into account that not all the rules grant a reward we use two types of rules: *regular rules* that have an execution cost and *granting rules* that have an execution cost and grant a reward. For example if the administrator objective is to secure her network, then firewalling a service (removing it from the `Public` set) will prevent it from being compromised but it is a temporary measure, and therefore should not grant a reward. At the opposite, patching the service (removing it from the `Vuln` set) is a permanent measure and grants a reward.

3.4 Player Rules

The set of rules used for the example focuses on intrusion and is meant to be very general. It is just an example to give the flavor of what is possible. It follows that the cost and time associated with each rules are meant to be in order of magnitude of what is commonly accepted but not necessarily accurate. The seven rules used in the example are shown in Figure 2.

1) **Pre** : $Vuln \wedge Public \wedge \neg Compr$
   $\Longrightarrow$ 2, I, Compromise 0day, 20000
   Effect : $Compr$
2) **Pre** : $Vuln \wedge Public \wedge \neg Compr$
   $\Longrightarrow$ (7, I, Compromise public, 5000)
   Effect : $Compr$
3) **Pre** : $\neg Compr \wedge \Diamond Compr$
   $\Longrightarrow$ (4, I, Compromise backward, 5000)
   Effect : $Compr$
4) **Pre** : $Compr \wedge \Diamond \neg Compr$
   $\Longrightarrow$ (4, I, Compromise forward, 5000)
   Effect : $\Diamond Compr$
5) **Pre** $Public \wedge Vuln$
   $\longrightarrow$ (1, A, Firewall, 10000)
   Effect $\neg Public$
6) **Pre** $\neg Public \wedge \neg Vuln \wedge NeedPub$
   $\longrightarrow$ (1, A, UnFirewall,0)
   Effect $Public$
7) **Pre** $Vuln \wedge \neg Compr$
   $\longrightarrow$ (3, A, Patch, 500)
   Effect $\neg Vuln \wedge \neg Compr$

Fig. 2. Set of rules

We take the convention that a `granting rule` uses the $\Longrightarrow$ double arrow and that a `regular rule` uses the $\longrightarrow$ single arrow. Rules `Compromise 0day`(1) and `Compromise Public`(2) say that if a vertex is vulnerable (`Vuln`), public (`Public`) and not compromised (`Compr`) then it can be compromised. The difference between the two is the time required to compromise the service (2 or 7 units) and the cost required (20000 or 5000). The use of these two rules allows to express that using a 0 day exploit over a public exploit provides an advantage in terms of time and a disadvantage in terms of cost. Accordingly the administrator `patch` rule (7) is slower than the `compromise 0day` rule and faster than the `compromise public` one. Theses three rules model the windows of vulnerability [18]. The rule `Compromise backward` says that the intruder can take advantage of a dependency relation to compromise a vertex that depends on a compromised one. The CTL modal operator [6] $\Diamond$ allows to speak about successor. Accordingly $\Diamond Compr$ means "*it exist a successor that is compromised*". This rule models attacks that exploit trust relationship. For example when a DNS server is compromised the intruder can use it to redirect clients to spoofed sites. Similarly the rule `Compromise forward` (4) says that the intruder can take advantage of a dependency relation to compromise the successor of a compromised vertex. In our DG example (figure 3.1) if the intruder is able to compromise the intranet server, he can look in its configuration files to steal database credential. The rule `Firewall` (5) says that if a service is vulnerable (`Vuln`) and Public (`Public`) it can be firewalled. The cost of the rule is very high (10000) compared to the patch rule cost (500) because firewalling a public service will indeed prevent the intruder to access it but also forbids legitimate access. Thus this action induces an activity disturbance and a possible financial loss. Notice the $\longrightarrow$ arrow of this rule that denotes that no reward is granted. Finally the rule `Unfirewall` (6) is used to make public services that are not vulnerable and need to be public (`NeedPub`).

3.5 Play example

The play used as an example (figure 3) is an intruder strategy to compromise the network. Due to space constraints, rules name have been truncated. Column `Ti` stands for time, `Pl` for players, `Act` for action, `Ta` for target, `S` for successor node, `Pa` for payoff and `C` for cost. Furthermore `I` is for intruder and `A` is for admin. Every strategy presented in this paper is the output result of our prototype using the DG, the initial mapping set, and the set of rules presented above along with various strategy objectives. Even if this example seems simple, it still cannot be analyzed by hand because this game configuration leads to 4011 distinct plays.

Ti	Pl	Act	Rule	Ta	S	Pa	C
0	I	choose	Comp 0 Day	4	$\perp$	-	-
0	A	choose	Firewall	4	$\perp$	-	-
1	A	**execute**	Firewall	4	$\perp$	0	10000
1	A	choose	Patch	4	$\perp$	-	-
2	I	`fail`	Comp 0 Day	4	$\perp$	0	20000
2	I	choose	Comp 0 Day	5	$\perp$	-	-
4	I	**execute**	Comp 0 Day	5	$\perp$	31	40000
4	I	choose	Comp For	5	6	-	-
4	A	**execute**	Patch	4	$\perp$	21	10500
4	A	choose	UnFirewall	4	$\perp$	-	-
5	A	**execute**	UnFirewall	4	$\perp$	21	10500
5	A	choose	Patch	5	$\perp$	-	-
8	I	**execute**	Comp For	5	6	1382	45000
8	I	choose	Comp Back	2	5	-	-

Ti	Pl	Act	Rule	Ta	S	Pa	C
8	A	**execute**	Patch	5	$\perp$	52	11000
12	I	`fail`	Comp Back	2	5	1382	50000
12	I	choose	Comp Back	4	6	-	-
16	I	**execute**	Comp Back	4	6	1403	55000
16	I	choose	Comp Back	1	4	-	-
20	I	**execute**	Comp Back	1	4	1404	60000
20	I	choose	Comp Back	2	4	-	-
24	I	**execute**	Comp Back	2	4	1405	65000
24	I	choose	Comp For	2	5	-	-
28	I	**execute**	Comp For	2	5	1436	70000
28	I	choose	Comp Back	3	5	-	-
32	I	**execute**	Comp Back	3	5	1437	75000

Fig. 3. Play example Intruder maximum payoff

The example is read as follow: At *time 0* the intruder chooses to use an 0 day exploit against the Web server (Target 4). At the same time the administrator start firewalling the Web server. Because firewalling is faster than exploiting the 0 Day vulnerability, the administrator is able to firewall the web server before the 0day exploitation is successful (*time 1*). The administrator starts to patch the web server. At *time 2* the intruder is taken by surprise by the administrator because the web server is firewalled before his exploitation is successful, hence the rule execution fails. He chooses to try another 0day exploit against the email server (target 5, *time 2*). At *time 4* the administrator has finished to patch the web server and decides to unfirewall it since it is no longer vulnerable. Meanwhile the intruder compromises the email server and decides to use his newly gained access to compromise the user database (target 6). At *time 5* the administrator decides to patch the email server (target 4). At *time 8* the intruder has compromised the user database (target 6). At the same moment the administrator has finished to patch the email server (node 5). Therefore at *time 12* the intruder fails to compromise the client 2 from the email server (Succ 4) because the email server is no longer vulnerable and compromised. However the intruder still has access to the database user server (node 6) and he uses this access to compromise the web server (*time 16*). From there he compromises the client 1 (*time 20*) and the client 2 (*time 28*). He uses his access on client 2 to compromise again the web server (node 5, *time 28*) and finally owns the network by compromising the client 3. As one can see the interaction between players leads to very complex plays even for this simple example. This play emphases that analyzing administrator and intruder interaction on the network cannot be achieved by hand.

3.6 Vertex Value Computation

The first type of vertex pricing that deserves attention is when the same value is used for every vertex. This is used to model the the question : ”What is the intrusion that will compromise the maximum number of services ?” because in this case each vertex have the same value,and therefore the intruder reward is maximized when the number of host compromised is maximized as well.

The second type of vertex pricing used is the one where vertices values are used to express which services are the most important for a given network. Theses values can be assigned by hand or can be inferred by an algorithm. While very interesting, studying the effectiveness of the various algorithms that can be used to compute theses values is out of the scope of this paper.
We currently use an algorithm inspired by the Google PageRank [27] one: each vertex value is the sum of its predecessor values weighted by dependency values.

Dependency value is based on its frequency because we claim that the more a dependency is solicited the more it is important for the network. While arguable our algorithm provides interesting results. The algorithm use the following formula to compute a vertex value:

$$value = \sum_{1}^{i} vertexValue(i) \times dependencyValue(i)$$

where i is the number of vertex predecessor, $vertexValue(i)$ is the function that return the value of the predecessor i and $dependencyValue(i)$ is the function that return the value of the dependency between the predecessor i and the vertex. This algorithm applied to our DG example (figure 3.1) with dependency frequency detailed in figure 3.6 give the result presented in figure 3.6. Frequency values were set by hand for example purpose only.

Dependency	Value
Client 1 $\longrightarrow$ Web server	10
Client 2 $\longrightarrow$ Web server	10
Client 2 $\longrightarrow$ Email server	15
Client 3 $\longrightarrow$ Email server	15
Web server $\longrightarrow$ database server	20
Email server $\longrightarrow$ database server	30

Vertex	Value
Client 1	1
Client 2	1
Client 3	1
Web server	21
Email server	31
Database server	1351

Fig. 4. Example of dependency values (left) and the resulting computed vertices values (right)

4 Strategy objective

In game theory a strategy is the optimal succession of actions (play) that a player can perfom to achieve his goal. As said previously, translating real world network security goals into reachability property is not expressive enough and error-prone .Therefore leveraging the cost and reward extension introduced in the previous section we introduce a new kind of analysis goal called **strategy objectives**, that combines symbolic and numerical objectives.

The logical formula is used to express which plays are acceptable strategies. The numerical objectives are used to select among these potential candidate, the one that fulfill the most players objectives. To the best of our knowledge this the first time that symbolic and numerical objectives are combined to express analysis goal. Strategy objectives allows to express naturally many network security. For example it allows to express that the goal of the administrator is to patch her network (logical formula) in minimum amount of time and for the lowest cost possible (numerical constraints). More formally we define strategy objectives as:

Definition 1 (Strategy objectives) *A set of strategy objectives is the tuple* $S : (name, \mathsf{P}, \mathcal{O}, \mathcal{R}, \varphi)$ *where* `name` *is the strategy name,* P *its owner,* $\mathcal{O}$ *is the set of numerical objectives,* $\mathcal{R}$ *is the numerical objectives priority strict order and,* φ *is the logical formula that a play needs to satisfy to be a valid strategy.*

4.1 Objectives

Strategy objectives $\mathcal{O}$ are assigned on play outcomes ϕ:

Definition 2 (Play outcomes) *is the unordered set of natural numbers* $\phi : \{\texttt{payoff}, \texttt{cost}, \texttt{opayoff}, \texttt{ocost}, \texttt{time}\}$ *where* `payoff` *is the player payoff ,* `cost` *is the player cost ,* `opayoff` *is the player opponent payoff,* `ocost` *is the opponent cost, and* `time` *is the play duration.*

The players P payoff for the play ρ is the sum of all the rewards granted by the successful execution of his granting rules. Rule reward is the value of the DG vertex targeted by the rule execution. The players P cost for ρ is the sum of all executed rule costs whether they are successful or not, because regardless of its success the player has invest the same amount of resource in it. It is convenient to describe numerical objectives by the concise language:

$$\begin{array}{rll} \mathcal{O} ::= & O & \text{Objective} \in \phi \\ \mid & \mathcal{O} \wedge \mathcal{O} & \\ \mid & MAX(O) & \text{maximize the value of objective O} \\ \mid & MIN(O) & \text{minimize the value of objective O} \\ \mid & O < x & x \in \mathbb{N} \\ \mid & O > x & x \in \mathbb{N} \end{array}$$

In this language, the patching strategy numerical objectives that seeks to minimize the time and the cost are written $MIN(Cost) \wedge MIN(Time)$. where the priority order can be either $\mathcal{R} : Cost > Time$ or $\mathcal{R} : Time > Cost$.

4.2 Objective logical formula

The objective logical formula is used to express which plays can be considered as valid strategy. In the patch strategy example, valid plays are those in which every vulnerable service is patched. An additional constraint can be that no services are compromised.

This constraint has two possible interpretations that lead to two very different results: first it can mean that at the end of the play no service is compromised but that at some point a service could have been compromised and restored. Secondly it can mean that no service is ever compromised during the play. The difference between the two interpretations is illustrated on the following example: The figure 5 is the strategy where at the end of the play no service is compromised and figure 6 is the strategy in which no service is ever compromised. We see that in the first strategy, during a brief moment the service 4 is compromised whereas in the second one no service are ever compromised.

Ti	Pl	Act	Rule	Ta	S	Pa	C
0	I	choose	Comp 0 Day	4	$\perp$	-	-
0	A	choose	Patch	4	$\perp$	-	-
2	I	**execute**	Comp 0 Day	4	$\perp$	21	20000
2	I	choose	Comp 0 Day	5	$\perp$	-	-
3	A	**execute**	Patch	4	$\perp$	21	500
3	A	choose	Patch	5	$\perp$	-	-
4	I	**execute**	Comp 0 Day	5	$\perp$	52	40000
4	I	choose	Comp For	5	6	-	-
6	A	**execute**	Patch	5	$\perp$	52	1000
8	I	`fail`	Comp For	5	6	52	45000

Fig. 5. A strategy where no service is Compromised at the end of the play

To express the second type of constraint the CTL [6] operator $\Box$ is needed. This operator is used to express that a constraint needs to be true for every state of the play. We also use the $\Diamond$ operator to express that a constraint need to be true at some point. This is handful for example in information leak intruder strategy to express that we seek to find a play where at some point a service was compromised. Thus the strategy formula is expressed in the following fragment

Ti	Pl	Act	Rule	Ta	S	Pa	C
0	I	choose	Comp 0 Day	4	$\perp$	-	-
0	A	choose	Firewall	4	$\perp$	-	-
1	A	**execute**	Firewall	4	$\perp$	0	10000
1	A	choose	Firewall	5	$\perp$	-	-
2	I	`fail`	Comp 0 Day	4	$\perp$	0	20000
2	I	choose	Comp 0 Day	5	$\perp$	-	-
2	I	choose	Comp 0 Day	5	$\perp$	-	-
2	A	**execute**	Firewall	5	$\perp$	0	20000
2	I	choose	Comp 0 Day	5	$\perp$	-	-
2	A	choose	Patch	4	$\perp$	-	-
4	I	`fail`	Comp 0 Day	5	$\perp$	0	40000
5	A	**execute**	Patch	4	$\perp$	21	20500
5	A	choose	UnFirewall	4	$\perp$	-	-
6	A	**execute**	UnFirewall	4	$\perp$	21	20500
6	A	choose	Patch	5	$\perp$	-	-
9	A	**execute**	Patch	5	$\perp$	52	21000
9	A	choose	UnFirewall	5	$\perp$	-	-
10	A	**execute**	UnFirewall	5	$\perp$	52	21000

Fig. 6. The administrator dominant strategy where services are never compromised (right)

of CTL:

$$\begin{array}{rll} \varphi ::= & A & \text{Atomic proposition} \\ | & \neg\varphi & \\ | & \varphi \wedge \varphi & \\ | & \forall A & \\ | & \exists A & \\ | & \Box\varphi & \\ | & \Diamond & \end{array}$$

We take the convention that if a constraint is specified without the $\Diamond$ and the $\Box$ operator the the constraint has to be true only on the last state of the play. In this fragment the patching strategy formula that ensures that no vertex is ever compromised (belongs to the set `Compr`) and that every vertex is not vulnerable at the end of the play is written:

$$\Box\neg Compr \wedge \neg Vuln$$

4.3 Dominant Strategy

The natural question that arises is what class of strategy objectives should be considered for network security. A naive idea would be to consider the class of objectives that minimizes/maximizes player cost/reward only and ensures by a set of constraints that the player goals are fulfilled. For example a patching strategy that minimizes the cost and ensures that no vertex is ever compromised and that every vertex is not vulnerable at the end of the play:

$$S : (patch, Admin, MIN(Cost), Cost, \Box\neg Compr \wedge \neg Vuln))$$

However this idea leads to find an incorrect strategy because the cost is minimal when the opponent makes "mistakes":

Ti	Pl	Act	Rule	Ta	S	Pa	C
0	I	choose	Comp Public	4	$\perp$	-	-
0	A	choose	Patch	5	$\perp$	-	-
3	A	**execute**	Patch	5	$\perp$	31	500
3	A	choose	Patch	4	$\perp$	-	-
6	A	**execute**	Patch	4	$\perp$	52	1000
7	I	`fail`	Comp Public	4	$\perp$	0	5000

Here the intruder could have been more effective. For example, he could have used an 0 day exploit at the beginning.

Thus a more interesting class of strategies to consider for network security is the one that minimize/maximize the cost/reward and is successful whatever the opponent does. They are the best set of actions against the worst case. Thus in our patching strategy example the administrator wants to have the less costly patching strategy that is effective even against the worst attack. This class of strategies is founded by adding an objectives that maximize/minimize the opponent cost/reward. They are commonly called strictly dominant strategies [31]. Dominant strategies are effective against the worst case but also every less effective case. In other word it means that when a player have a strictly dominant strategy and he performs it, he is ensured to fulfill his objective regardless of what his opponent do. The notion of unbeatable strategy was introduced in biology for evolution dynamic [13]. From the network security perspective it means that when the administrator have a dominant strategy for a given set of goals whatever the intruder knowledge of the network has and whatever mistake he makes, this strategy ensure her that she will always accomplish her goals. In the example a dominant strategy exists for the administrator (figure 6). When used it ensure that the administrator, regardless of the intruder actions, will be able to path the two vulnerable services before any services is compromised.

Dominant strategies does not always exists. For instance if the administrator can't firewall her public services then the intruder has a dominant strategy that uses 0 day exploit (Figure 8). This means that regardless of administrator action, the intruder can always compromise services, as long as he use 0day exploit. On the other hand the administrator has a weakly dominant strategy that is able to dominate every intruder strategy that does not use 0 day exploit (Figure 7). This weakly dominant strategy is the best strategy that the administrator can come up with in this situation. Its the strategy that will minimize the number of her opponent successful strategies. In other word, this is the strategy that will allow her to defeat intruder with partial knowledge of the network or that made mistake.

Ti	Pl	Act	Rule	Ta	S	Pa	C
0	I	choose	Comp 0 Day	4	⊥	-	-
0	A	choose	Patch	5	⊥	-	-
2	I	**execute**	Comp 0 Day	4	⊥	21	20000
2	I	choose	Comp For	4	6	-	-
3	A	**execute**	Patch	5	⊥	31	500
3	A	choose	Patch	4	⊥	-	-
6	I	**execute**	Comp For	4	6	1372	25000
6	I	choose	Comp Back	1	4	-	-
6	I	choose	Comp Back	1	4	-	-
6	A	**execute**	Patch	4	⊥	52	1000
10	I	`fail`	Comp Back	1	4	1372	30000
10	I	choose	Comp Back	4	6	-	-
14	I	**execute**	Comp Back	4	6	1393	35000
14	I	choose	Comp Back	1	4	-	-
18	I	**execute**	Comp Back	1	4	1394	40000
18	I	choose	Comp Back	2	4	-	-
22	I	**execute**	Comp Back	2	4	1395	45000
22	I	choose	Comp For	2	5	-	-
26	I	**execute**	Comp For	2	5	1426	50000
26	I	choose	Comp Back	3	5	-	-
30	I	**execute**	Comp Back	3	5	1427	55000

Fig. 7. Incident dominant strategy when the firewall rule is unavailable

4.4 Complexity

We now study the decidability of finding the best strategy for given set of strategy objectives . The key issue is that plays can be infinite. However they are ultimately periodic paths because an AG is finite and therefore even when a game has an infinite play it is possible to decide which play is the best for a

Ti	Pl	Act	Rule	Ta	S	Pa	C
0	I	choose	Comp Public	5	$\perp$	-	-
0	A	choose	Patch	5	$\perp$	-	-
4	A	**execute**	Patch	5	$\perp$	31	500
4	A	choose	Patch	4	$\perp$	-	-
7	I	`fail`	Comp Public	5	$\perp$	0	5000
7	I	choose	Comp 0 Day	4	$\perp$	-	-
8	A	**execute**	Patch	4	$\perp$	52	1000
10	I	`fail`	Comp 0 Day	4	$\perp$	0	25000

Fig. 8. Administrator weakly dominant strategy when the firewall rule is unavailable

given set of objectives. To decide so , two things must be known: the play outcome and if the play satisfies objectives formula. For the play outcome we have the following result (proof is in App. A):

Lemma 1. *An infinite play outcome can be computed by examining a short finite prefix.*

For formula satisfiability we have the following result (proof is in App. B):

Lemma 2. *For an infinite play the decidability of objectives formula satisfaction can be reduced to verifying the formula on a short finite prefix.*

This lead us to the central theorem for decidability (proof is in App. C):

Theorem 1 *Deciding if a play is the one that fulfill the most strategies objectives is decidable for any play by looking at a finite number of states.*

Moreover we can prove that deciding if a play satisfies the most strategy objectives can be done in polynomial time (proof in App D):

Theorem 2 *Deciding if a play satisfies the most strategy objectives can be decided in polynomial time* $\mathsf{O}(s \times |\varphi|)$ *where* s *is the number of states in the finite prefix and* φ *the formula to verify.*

Ultimately we have the general decidability theorem (Proof is in App. E):

Theorem 3 (Decidability) *Finding the strategy that fulfill the most strategies objectives over an anticipation game is decidable.*

4.5 Memory Space complexity

A key property for implementation is that the memory required to find a strategy for a given set of objectives is linear (proof is in App F).:

Theorem 4 *Memory space complexity worst case can is:* $WC = Set \times V \times R \times (S + 1)$ *Where Set is the finite memory space required to hold sets mapping values, V is the number of vertices of the dependency graph, R is the number of rules and S is the number of strategies researched.*

5 Evaluation

In order to evaluate that AG can be used in practice, we have conducted a set of evalution with our prototype on a standard Intel 2.9GHz core 2 Linux. Each benchmark was run three times and the reported time is the mean. Two set of benchmarks have been conducted. The first set was used to determine is the AG framework is usable in practice. The second set was used to measure the impact of strategy on the analyzer performance and ensure that they are consistent with theoretical bounds.

The first set of benchmark was done by running the analyzer against a large example. This example use the set of rules presented in the Sec. 3.4 and an initial random network state and look for the intrusion and defense strategy presented in Sec 4. The random initial state is composed of 5200 nodes, 27000 dependencies and 3 random vulnerabilities. Each of the 200 servers nodes have 10 random dependencies and each of the 5000 clients nodes have 5 random dependencies. To ensure that the generated initial state is not a degenerate case, we have used 10 different initial states. The analyzer have the same performance regardless of the initial states used. In order to deal with such large example, our analyzer uses numerous optimization including a static analysis of the dependency graph shape and rules set based on the strategy constraints. Theses optimization, in particular the early cut optimization, explain why it has been possible to find the optimal strategy for the defense objectives and only an approximate strategy for the intrusion one. The approximate strategy found is sound and not complete. Which mean that it is a valid strategy but there is no guarantee that it is the best possible as it could be a local optimum. However tests on smaller example suggest that it is often the optimum one. Benchmark results presented in figure G, tshow that in practice, AG can be used to analyze a new situation even for a complex network in a matter of seconds.

Nb Nodes	Nb Dep	Strategy	type	Time
5200	27000	Defense	Exact	2.4 sec
5200	27000	Intrusion	Approximate	55 sec

Fig. 9. Analyzer performance benchmark

We evaluate the impact of strategy on analysis performance by conducting two type of benchmark. The first was designed to measure the impact of strategy on analyzer speed and the second on memory usage. In order to have the most accurate evaluation possible, all analyzer optimizations were disabled. The game we choose as a baseline for our benchmark is an expended version of example presented in the paper with ten additional clients. Without optimization, adding clients increase drastically the interleaving generated by the rules `compromise forward` and `compromise backward`.
For the speed performance benchmark, we have used as a baseline the time required by the analyzer to run every possible plays generated by the game without strategy. Then we have run the analyzer with an increasing number of strategy requests.We have requested a number of strategies that range between 0 and 100. Every strategy requested had $\Box$ constraint to be in the worst case possible: Strategy are evaluated on every states of the play. Experimentation results (see diagram in App. G) show that that the time requested to analyze the game grows linearly in the number of strategies. Which is consistent with the theorem 2.

Secondly we use the memory profiler **massif** from the **valgrind** tool suite [24] to verify that the memory needed by the analyzer grows linearly in the number of strategies as proved in theorem 4. We run the analyzer with a growing number of strategies ranging from 1 to 30. For each test we take the memory peak reported in massif diagram as (in appendix H). In this figure the memory is almost constant because the optimal strategies were found early in the game. Results, visible in the figure on App G, show that as expected the memory needed is linear in the number of strategies which is consistent with thetheoretical bound proved in Sec 4.4.

6 Conclusion

In this paper we have introduced strategies for anticipation games. We have shown that using strategy objectives as analysis goal allow to find answers to key security questions such as ”What is the best patching strategy in term of time and cost”. We have explained how our extension takes into account the financial dimension of the network security, making the AG the first framework that deals with time and the financial dimension of attack at the same time. We have proved that finding the strategy that fulfill the most strategy objectives is decidable and that it only required a linear memory space. We have also proved that verifying the validity of a strategy against a given play can be done in polynomial time. Finally we have evaluated the suitability of AG with strategies for practical use by fully implementing AG with strategy in a prototype written in C. Our evaluation shows that our prototype is capable of finding strategy for large network. Future work involves extending strategies with non-determinism to model attackers with various levels of knowledge and skill.

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. 5
2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. 5
3. M. Artz. *NetSPA : a Network Security Planning Architecture*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science., 2002. 4
4. M. R. B. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997. 5
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. 4
6. B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001. 8, 13
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. 1, 5
8. 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. 4
9. 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. 4
10. 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. 4
11. 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. 5
12. 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. 2, 5
13. W. D. Hamilton. Extraordinary sex ratios. *Science*, 1967. 15
14. T. Henzinger and V. Prabhu. Timed alternating-time temporal logic. In *Formats 06*, volume 4202, pages 1–18. Springer-Verlag, 2006. 5
15. 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. 4

16. 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. 4
17. 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. 5
18. 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. 8
19. K.-w. Lye and J. M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005. 5
20. 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. 5
21. 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. 5
22. N. Markey and P. Schnoebelen. Model checking a path. In R. M. Amadio and D. Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 251–265, Marseilles, France, Aug. 2003. Springer. 24
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. 4
24. N. Nethercote, R. Walsh, and J. Fitzhardinge. Building Workload Characterization Tools with Valgrind. *Workload Characterization, 2006 IEEE International Symposium on*, pages 2–2, 2006. 19
25. 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. 4
26. 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. 4
27. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998. 10
28. 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. 4
29. 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. 5
30. C. Ramakrishan and R. Sekar. Model-based analysis of configuration vulnerabilities. In *Journal of Computer Security*, volume 1, pages 198–209, 2002. 4
31. E. Rasmusen. *Games and Information*. Blackwell publishing, 2007. 5, 15
32. 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. 3, 4

33. 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. 4
34. B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's journal*, Dec. 1999. 4
35. B. Schneier. *Secrets & Lies: Digital Security in a Networked World.* Wiley, 2000. 4
36. 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. 5
37. H. R. Shahriari and R. Jalili. Modeling and analyzing network vulnerabilities via a logic-based approach. 4
38. 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. 4
39. L. P. Swiler. A graph-based network-vulnerability analysis system. In *New Security Paradigms Workshop*, pages 71 – 79. ACM Press, 1998. 4
40. 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. 4
41. 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. 4
42. 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. 4

A Proof of Lemma 1

Proof. The number of distinct states for any play is finite because the number of rules and DG vertices are finite. Therefore an infinite play has a finite number of distinct states. Therefore if it is an infinite play, it has a loop. Payoff and cost functions are monotonic, therefore outcome values can only increase. Consequently there are two possible cases for a given outcome value. One: if the value is incremented during the first loop iteration, it will be incremented at every loop iteration and therefore the value diverges. Two: the value remains the same after the first loop iteration. In this case the value will remain the same regardless the number of iterations. Therefore it is possible to compute the outcome of an infinite play in a short finite prefix.

B Proof of Lemma 2

Proof. As explained in lemma 1, infinite plays have a loop. Therefore at the end of the first iteration all the distinct states of the play have been enumerated. Hence if objectives formula holds at the end of the first iteration, it holds for any number of iterations. Therefore it is possible to compute the validity of an infinite play in a short finite prefix.

C Proof for Theorem 1

Proof. It is trivial for finite play and for infinite play it is true because both strategy formula satisfiability and strategy outcome can be verified on a short prefix by lemma 1 and lemma 2.

D Proof of Theorem 2

Proof. By lemma 2 we know that the number of states that need to be verified is finite. Moreover since play constraint are expressed in CTL, the theorem 3.1 of [22] holds. This theorem states that path model checking for CTL can be done in PTIME if the number of states is finite.
Additionally by lemma 1 we know that the the outcome of the play is known after examining the same short prefix used to determine if the formula is satisfied. Moreover comparing the outcome of the play with the previous selected play is done in constant time. Therefore deciding which play is the best can be done in polynomial time

E Proof of Theorem E

Proof. The number of dependency graph vertices, sets and rules are finite. Therefore the number of distinct states is finite. Hence the number of plays is finite. By theorem 1, every play is decidable by examining a finite number of states. Therefore the number of states to examine to find a strategy is finite. Additionally by hypothesis the number of strategies is finite hence each state needs to be inspected a finite number of times. Therefore finding a strategies that fulfill strategy objectives over an AG is decidable.

F Proof of Theorem 4

Proof. The longest possible finite play is the one where every rule is executed against every dependency graph vertex twice because a rule execution can either fail or be successfully against a given vertex. By theorem 1 even for infinite plays it is sufficient to store this number of states because adding another state is equivalent to adding the first state of the second loop iteration which is not needed. Therefore the worst memory case occurs when every strategy play and the current play are equal to the longest play.

G Performance evaluation diagram

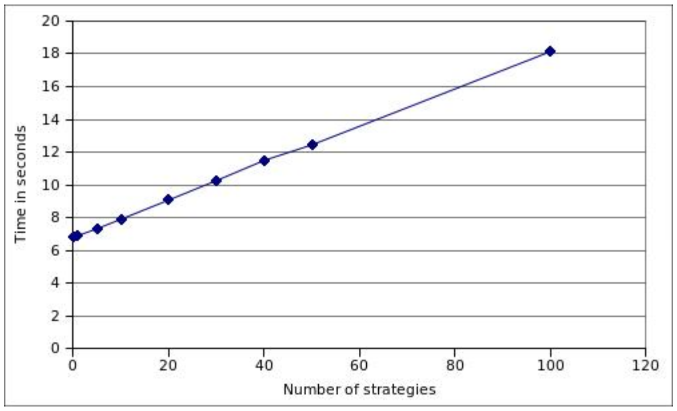

Fig. 10. Strategies impact on NetQi speed)

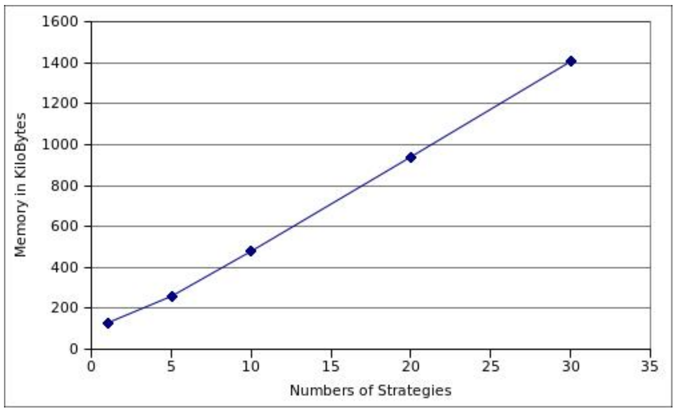

Fig. 11. memory usage

H Massif memory analysis output

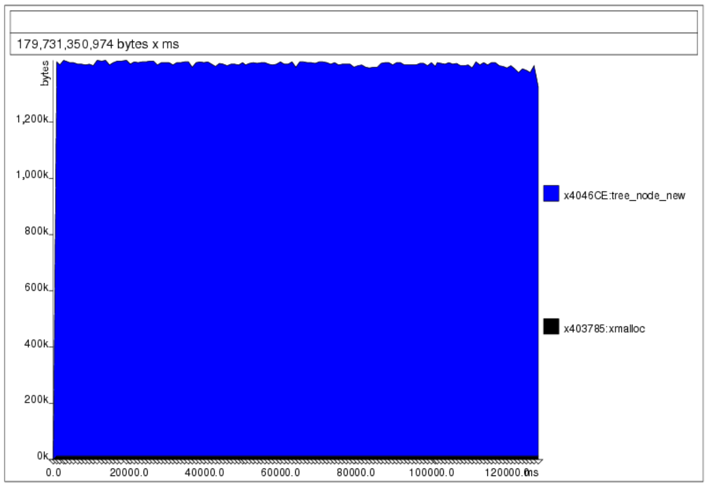

Fig. 12. Analyzer memory consumption for the execution with 30 strategies reported by Massif the Valgrind tool