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, INRIA, CNRS
[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, expressing intruder goal as a safety property is tedious and error prone on large networks because it assumes that the analyst has prior and complete knowledge of critical network services and knows what the attacker targets will be.
In this paper we address this issue by introducing a new kind of goal called "*strategy objectives*". Strategy objectives mix 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, and prevention. Anticipation games (AG) [4] 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 player's actions.

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 where 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 preconditions and postconditions written in a modal logic. Describing the model only with the network initial state and a set of rules relieves the security analyst from the tedious and error prone burden of explicitly describing each network state 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's initial state [3]. 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.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*" [7], 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 take into account network topological information such as dependency between network services, which allow them to model *collateral effects*. For example when a DNS server is unavailable due to a DDOS then by collateral effect, the web server is merely available because browsers can't perform DNS resolution.

Although using AG to analyze attacks provides a substantial improvement over standard attack graphs, there is still one side of attack modeling that remains tedious and error-prone: defining the analysis goal. So far as standard attack graph[16], the current AG analysis goal is to prove that a given safety property holds for a given model. However, network analysis 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 a primary security objective, especially when working on large networks. 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 all of 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 have been combined to express security goals. Note that being able to select the most relevant candidate 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 answer a brand new range of question that more closely match administrator and security analysts needs. For example, using strategy objectives it is possible to answer the question: "*What is the most effective patching strategy in terms of cost or time ?*". Finally the introduction of action costs and rewards takes into account the financial dimension of attacks which is a central concern of network attacks. Taking into account action cost allows us to reason about the costs required to launch an attack, the loss induced by it, and the investment required to prevent it.

Our main contribution is the extension of AG with strategy objectives. This extension allows the analysis to answer key network security questions and to capture the financial dimension of the attack.
As far as we know, our extension the AG framework is the first attack model that covers both the financial and temporal aspects 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. The evaluation performed with our prototype shows that in practice this framework can be used to find strategy for large networks (Thousands of nodes). This evaluation also demonstrates that practical results are consistent with the theoretical bounds we have proven.

The reminder of this paper is organized as follows. In Sect. 2, we will survey related work. 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 the impact of using strategy in term of speed and memory with our prototype. We show that our experiments are consistent with the theory and that strategies can be used in practice.

2 Related Work

Model checking for attack graphs was introduced by Ammann and Ritchey [16]. It is used to harden security [13]. Various methods have been proposed for finding attack paths, i.e., sequences of exploit state transitions, including logic-

based approaches [17] and graph-based approaches [12].Research has also been conducted on formal languages to describe actions and states in attack graphs [5].Anticipation games 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]. The notion of cost for attack appears in [6]. Mahimkar and Shmatikov have used the game theory to model denial of service in [11] The use of games for network security was introduced by Lye and Wing [10]. The Anticipation Game framework was introduced in [4]. A dedicated model-checker called NetQi [3] has been developed to accommodate anticipation game specificities.

3 Anticipation Games with Cost and Rewards

This section briefly recalls what an *anticipation game* (AG) is and explains the extension made to introduce strategy in the model. Intuitively, an AG can be represented as a graph. Each node of the graph describes the network state at any 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 a *Dependency Graph* (DG) and a finite set of states. DGs are meant to remain fixed over time and describe the relation between services and files. The figure 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 dependencies between them. In the example below, 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.

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,

States	1	2	3	4	5	6
ρ(Vuln)	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
ρ(Public)	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
ρ(Compr)	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
ρ(NeedPub)	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\bot$

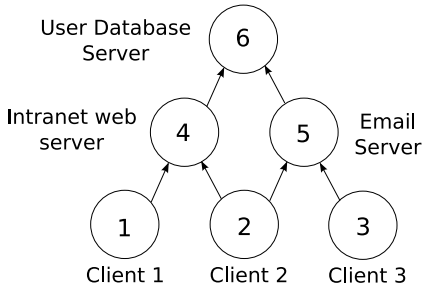

Fig. 1. The intial network state (left) and the dependency graph (right)

and are public because (ρ(NeedPub)), ρ(Public), and (ρ(Vuln)) return true ($\top$). It also indicates that no vertex is compromised as ρ(Compr) returns false ($\bot$) for every vertex. Finally the set ρ(NeedPub) is used by the `Unfirewall` rules to know which vertices should be made public.

3.2 Player's 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 :

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

This 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 this rule to the *Email server* then it is required that the preconditions are fulfilled when he chooses to apply it but also after the 30 units of time required to execute it because the network state might have changed during this time due to administrator actions. For example, the administrator could 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 with costs and rewards. The natural way to do so is to add a cost to rules and an a reward to each DG vertex.

Costs are added to rules because it is obvious that some actions 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 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 the fact 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 end of the spectrum, patching the service (removing it from the `Vuln` set) is a permanent measure and grants a reward. Note that computing the value of network assets is a topic by itself [14] and we we assume that one of the existing methods is used to compute the rewards associated to vertices.

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 meant to give a flavor of what is possible with our model. It follows that the cost and time associated with each rule are meant to be on the order of magnitude of what is commonly accepted, but these measures are not necessarily completly accurate. The seven rules used in the example are shown in Figure 2.

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 us to express the fact that using a 0 day exploit instead of a public exploit provides an advantage in terms of time and a disadvantage in terms of cost. To be consistent with this idea, the administrator `patch` rule (7) is

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

slower than the `compromise 0day` rule and faster than the `compromise public` one. These three rules model the windows of vulnerability [9]. 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 modal operator [2] $\Diamond$ allows preconditions and postconditions to speak about vertice successor. For example $\Diamond Compr$ means "*there exists a successor that is compromised*". This operator is used to model attacks that exploit trust relationships and collateral effects such as the attack where a compromised DNS server is used 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 1) if the intruder is able to compromise the intranet server, he can look in its configuration files to steal database credentials. 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 that aimed at compromising 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 NetQi 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	⊥	-	-
0	A	choose	Firewall	4	⊥	-	-
1	A	**execute**	Firewall	4	⊥	0	10000
1	A	choose	Patch	4	⊥	-	-
2	I	fail	Comp 0 Day	4	⊥	0	20000
2	I	choose	Comp 0 Day	5	⊥	-	-
4	I	**execute**	Comp 0 Day	5	⊥	31	40000
4	I	choose	Comp For	5	6	-	-
4	A	**execute**	Patch	4	⊥	21	10500
4	A	choose	UnFirewall	4	⊥	-	-
5	A	**execute**	UnFirewall	4	⊥	21	10500
5	A	choose	Patch	5	⊥	-	-
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	⊥	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 figure 3 example is read as follows: At *time 0* the intruder chooses to use an 0 day exploit against the Web server (Target 4). At the same time the administrator starts 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 patching 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 patching 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 the web server again (node 5, *time 28*) and finally owns the network by compromising the client 3. This play illustrates that the interaction between players leads to very complex plays even when the initial situation is simple. This emphases that analyzing administrator and intruder interactions on a real network cannot be achieved by hand.

4 Strategy objectives

In game theory, a strategy is the optimal succession of actions (plays) that a player can perform to achieve his goal. As said previously, translating real world network security goals into reachability properties is not expressive enough and is also error-prone. Therefore in this section, we introduce a new kind of analysis goal called **strategy objectives** that combine symbolic constraints and numerical objectives by leveraging the notion of cost and reward introduced previously.

Symbolic constraints, expressed as a CTL logical formula, are used to express which plays are acceptable strategies. Numerical objectives are used to select among these potential candidates, the one that fulfill the most player interests. To the best of our knowledge this the first time that symbolic and numerical objectives have been combined to express analysis goal. Strategy objectives allow security analysts to express naturally many network security goals. For example it allow security analysts 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 *is 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 Numerical Objectives

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

Definition 2 (Play outcomes) *are 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 duration of the play.*

The player P payoff for the play ρ is the sum of all the rewards granted by the successful execution of his granting rules. A 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 invested the same amount of resource in it. A strategy numerical objective is either the maximization or the minimization of one of these play outcome. For instance, an administrator might want to find a patching strategy that *minimizes* the *cost* and the *time*.

4.2 Symbolic Constraints

Symbolic constraints are used to express which plays can be considered valid strategies. 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 that with the first interpretation, having a service compromised for a brief moment is acceptable, whereas with the second interpretation, it is not.

To express the second type of constraint the CTL [2] operator $\Box$ is needed. This operator is used to express the fact that a constraint needs to be true for every state of the play. We also use the $\Diamond$ operator to express the fact that a constraint needs to be true at some point. This operator is used, for example to express information leak strategy where at some point a service was compromised. Thus the strategy formula is expressed in the following fragment 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 then this constraint has to be true only on the last state of the play. Patching strategy symbolic constraints are used to ensure that no vertex is ever compromised (belongs to the set `Compr`) and that every vertex is not vulnerable at the end of the play are written: $\Box\neg Compr \wedge \neg Vuln$ in our CTL fragment.

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. The following patching strategy objectives minimize the cost and ensures that no vertex is ever compromised and that every vertex is not vulnerable at the end of the play is:

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

However these strategy objectives lead to 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

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 ones that minimize/maximize the cost/reward and are successful whatever the opponent does. These strategies are the best set of actions against the worst case. Thus, in our patching strategy example, the administrator wants to have the least costly patching strategy that is effective even against the worst case attack. This class of strategies is computed by adding objectives that maximize/minimize the opponent's cost/reward. They are commonly called strictly dominant strategies [15]. Dominant strategies are effective against the worst case but also every less effective case. In other words, when a player has a strictly dominant strategy and he performs it, he is able to fulfill his objective regardless of what his opponent does. From the network security perspective, it means that when the administrator has a dominant strategy

for a given set of goals, whatever her opponent do, this strategy will ensure her that she will always accomplish her objective. In the example, a dominant strategy exists for the administrator.When used, it ensures that the administrator, regardless of the intruder actions, will be able to patch the two vulnerable services before any service is compromised.Note that in the general case dominant strategies do not always exist.

4.4 Complexity

We now study the decidability of finding the best strategy for a 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 given set of objectives. To decide so, two things must be known: the play outcome and if the play satisfies the logic formula used to express the symbolic constraints. For the play outcome we have the following result:

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

For formula satisfiability we have the following result:

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 leads us to the central theorem for decidability:

Theorem 1 *Deciding if a play is the one that fulfills the most strategy 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:

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 *is the formula to verify.*

Ultimately we have the general decidability theorem:

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

A key property for implementation is that the memory required to find a strategy for a given set of objectives is linear:

Theorem 4 *Memory space complexity worst case 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 evalutions 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 sets of benchmarks have been conducted. The first set was used to determine if 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 benchmarks was done by running the analyzer against a large example. This example uses the set of rules presented in the Sec. 3.4, an initial random network state, and looks 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 server nodes has 10 random dependencies and each of the 5000 client nodes has 5 random dependencies. To ensure that the generated initial state is not a degenerate case, we used 10 different initial states. Our prototype has the same performance regardless of the initial states used. In order to deal with such a large example, our analyzer uses numerous optimization tricks, including a static analysis of the dependency graph shape and a static analysis of the rules set based on the shape of the symbolic constraints [3]. Benchmark results presented in figure 4 show 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. 4. Analyzer performance benchmark

We evaluate the performance impact of using strategy objectives instead of verifying a security property by conducting two types of benchmark. The

first type was designed to measure the impact of strategy on analyzer speed and the second to measure 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 expanded version of the example presented in this paper with ten additional clients. Without optimization, adding clients increases drastically the interleaving generated by the rules `compromise forward` and `compromise backward`. For this speed performance benchmark, we used as a baseline the time required by the analyzer to run every possible play generated by the game without strategy. Then we ran the analyzer with an increasing number of strategy requests. The requested number of strategies ranges between 0 and 100. Every strategy symbolic objectives contains a $\Box$ constraint to ensure that we are in the worst case possible: Our prototype has to evaluate the symbolic constraints satisfaction on every state of every play. Experimental results show that that the time requested to analyze the game grows linearly in the number of strategies, which is consistent with theorem 2. Secondly, we have used the memory profiler **massif** from the **valgrind** tool suite to verify that the memory needed by the analyzer grows linearly in the number of strategies as proved in theorem 4.

6 Conclusion

In this paper we have introduced strategies for anticipation games. We have shown that using strategy objectives as analysis goals allow us to find answers to key security questions such as "What is the best patching strategy in term of time and cost". We have described 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 fulfills the most strategy objectives is decidable and that it only requires a linear memory space. Finally, we have demonstrated the suitability of AG with strategies for practical uses by fully implementing AG with strategies in our prototype . Future work involves extending strategies with non-determinism to model attackers with various levels of knowledge and skill.

References

1. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
2. 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.
3. E. Bursztein. NetQi: A model checker for anticipation game. In M. Kim and M. Viswanathan, editors, *Proceedings of the 6th International Symposium on Automated*

Technology for Verification and Analysis (ATVA'08), Lecture Notes in Computer Science, Seoul, Korea, Oct. 2008. Springer.
4. 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.
5. 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.
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. 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.
10. K.-w. Lye and J. M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005.
11. 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.
12. 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.
13. 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.
14. 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.
15. E. Rasmusen. *Games and Information*. Blackwell publishing, 2007.
16. 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.
17. 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.