Elie Bursztein

Network Administrator and Intruder Strategies

Research Report LSV-08-02

February 2008

Network Administrator and Intruder Strategies

Elie Bursztein

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

Abstract. The anticipation game framework is an extension of attack graphs based on game theory. It is used to anticipate and analyze intruder and administrator interactions with the network. In this paper we extend this framework with cost and reward in order to analyze and find player strategies. Additionally this extension allows to take into account the financial aspect of network security in the analysis. Intuitively a strategy is the best succession of actions that the administrator or the intruder can perform to achieve his objectives. Player objectives range from patching the network efficiently to compromising the most valuable network assets. We prove that finding the optimal strategy is decidable and only requires a linear memory space. Finally we show that finding strategy can be done in practice by evaluating the performance of our analyzer called NetQi.

1 Introduction

Today's computers run a multitude of services, share trust relationships with numerous devices, and share various kinds of connectivity with other computers. This type of heavy interaction can lead to unforeseen vulnerabilities. The anticipation game (AG) framework [8] was created to analyze theses interactions. The AG framework is an extension of attack graphs [17] based on game theory. More specifically it uses a network oriented variant of the timed alternating-time temporal logic (TATL) [14]. This framework offers three key advantages over standard attack graph. First it takes into account dependency between services to detect *collateral effects*. For example if a company has all its DNS servers under a DDOS attack, by collateral effect its web site and its email server are unreachable because DNS name resolution fails. Secondly it models the dynamic interaction between the administrator, the intruder, and the network. For example the administrator interacts by patching a service whereas the intruder interacts by exploiting a vulnerability. Finally it is a timed model and therefore allows to take into account the time required to perform an action. This allows to capture security time issues such as the notion of windows of vulnerability[19].

In this paper we extend anticipation game with strategies. A strategy is the play that satisfy a set of constraints, such as no service is ever compromised during the play, and maximize/minimize players objectives such as execution time or actions cost. Intuitively finding a strategy can be intuited as finding the most efficient play among all the game plays that satisfy the requested set of constraints. Using strategy as analysis goal instead of the standard attack graph reachability property is more natural because the notion of strategy provide a natural answer to crucial network security questions such as what is the most efficient patching strategy given the situation or what is the most effective attack against my current network setup.

Moreover using cost and reward allows to model the cost and the financial impact of an intrusion which is not taken into account in standard attack graph. For instance, combined with the notion of time, this extension allows to model that a 0 day exploit provides a significant advantage to the intruder because it can be used before the corresponding patch is released but for a significantly higher cost than a public exploit.

The main contribution of this paper is the introduction of memoryless strategies for anticipation games. We prove that finding strategy is decidable and only requires a linear memory space. We also prove that verifying the validity of a strategy for a given play can be done in polynomial time even when play are infinite. Finally we have implemented AG with strategy in an open-source tool called NetQI [7] to evaluate the usability of strategies for practical use.

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 detail the game example that is used as a guideline for the rest of the paper. Sect. 4 details how strategies 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 [36, 37] and Kuang and al [45]. Model checking for attack graphs was introduced by Ammann and Ritchey [34]. They are used to harden security [27]. Various methods have been proposed for finding attack paths, i.e., sequences of exploit state transitions, including logic-based approaches [32, 40, 17, 39], and graph-based approaches [45, 41, 26]. Researchs have also been conducted on formal languages to describe actions and states in attack graphs [12, 42]. Some rely on grammars [43], some have a more practical focus [11], or specialize on IDS alert correlation

[24]. Some authors propose techniques that allow attack graphs to scale to large networks [16, 28]. Security metrics [30] have been developed, and supporting tools such as NetSPA [3] or Sheyner's tool [40] now exist.

The SIR model, which is similar to the compromising recovery cycle, is used to study the propagation of epidemics in biology [10]. Biological models for computer security were proposed recently [35]. As in computer virus propagation research [5, 44], biological models are an inspiration of anticipation games. The antibody (administrator) fights the disease (Intruder) to maintain the body alive (the network). Following this intuition, using games to capture this fight interaction appears natural.

Games have become a central modeling paradigm in computer science. In synthesis and control, it is natural to view a system and its environment as players of a game that pursue different objectives [9, 31]. In our model, the intruder attempts at causing the greatest impact on the network whereas the administrator tries to reduce it. Such a game proceeds for an infinite sequence of rounds. At each round, the players choose actions to play, e.g., patching a service, and the chosen actions determine the successor state. For our anticipation games we need, as in any *real-time* system, to use games where time elapses between actions [22]. This is the basis of the work on timed automata, timed games, and timed alternating-time temporal logic (TATL) [15], a timed extension to alternating-time Kripke structures and temporal logic (ATL) [2]. The TATL framework was specifically introduced in [14].

Timed games differ from their untimed counterpart in two essential ways. First, players have to be prevented from winning by stopping time. More important to us is that players can take each other by *surprise*: imagine that the administrator attempts to patch a vulnerable service, and this will take 5 minutes, it may happen that intruder is in fact currently conducting an attack, which will succeed in 5 seconds, nullifying administrator action. Second (this is allegedly more technical), a player cannot win by preventing time from diverging, i.e., from eventually tending to infinity [38]. Average reward games considered in TATL framework are considered in [1], but with the time move duration restricted to either 0 or 1. ATL was also extended to simply timed concurrent game structure [18]. Game strategies have been used to predict players actions in numerous domains ranging from economy to war [4, 33].

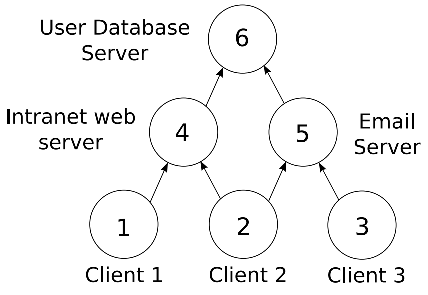
The notion of cost for attack appears in [13]. The use of games for network security was introduced by Lye and Wing [20]. The anticipation game framework was presented by Bursztein and Goubault-Larrecq [8]. Finally Mahimkar and Shmatikov used the game theory to model denial of service in [21].

3 Anticipation Games with Cost and Rewards

This section briefly recalls what an anticipation game is and details 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 edges represent the set of actions that both players, the administrator and the intruder, can perform to alter the network state. For example 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 1 presents the DG used as 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
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. A dependency graph and its associated sets of states mapping

The set of *states* 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 $\mathbf{Pre}\ F \overset{\Delta,p,a,c}{\longrightarrow} P$ 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

To have strategies AG needs to be extended to have action cost and reward. Action cost is added to rules and action reward is added by giving a reward value to each DG vertex. 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 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:

1 $\mathbf{Pre}\ Vuln \wedge Public \wedge \neg Compr \xRightarrow{(2,I,Compromise\ 0day,20000)} Compr$

2 $\mathbf{Pre}\ Vuln \wedge Public \wedge \neg Compr \xRightarrow{(7,I,Compromise\ public,5000)} Compr$

3 $\mathbf{Pre}\ \neg Compr \wedge \Diamond Compr \xRightarrow{(4,I,Compromise\ backward,5000)} Compr$

4 $\mathbf{Pre}\ Compr \wedge \Diamond \neg Compr \xRightarrow{(4,I,Compromise\ forward,5000)} \Diamond Compr$

5 $\mathbf{Pre}\ Public \wedge Vuln \xrightarrow{(1,A,Firewall,10000)} \neg Public$

6 $\mathbf{Pre}\ Public \wedge \neg Vuln \wedge NeedPub \xrightarrow{(1,A,UnFirewall,0)} Public$

7 $\mathbf{Pre}\ Vuln \wedge \neg Compr \xRightarrow{(3,A,Patch,500)} \neg Vuln \wedge \neg Compr$

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 [19]. 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 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 example (figure 2) is an intruder strategy to compromise the network. 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 can't be done by hand because this game configuration leads to 4011 distinct plays.

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

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	Intruder	choose	Compromise 0 Day	4	$\perp$	-	-
0	Admin	choose	Firewall	4	$\perp$	-	-
1	Admin	**execute**	Firewall	4	$\perp$	0	10000
1	Admin	choose	Patch	4	$\perp$	-	-
2	Intruder	`fail`	Compromise 0 Day	4	$\perp$	0	20000
2	Intruder	choose	Compromise 0 Day	5	$\perp$	-	-
4	Intruder	**execute**	Compromise 0 Day	5	$\perp$	31	40000
4	Intruder	choose	Compromise Forward	5	6	-	-
4	Admin	**execute**	Patch	4	$\perp$	21	10500
4	Admin	choose	UnFirewall	4	$\perp$	-	-
5	Admin	**execute**	UnFirewall	4	$\perp$	21	10500
5	Admin	choose	Patch	5	$\perp$	-	-
8	Intruder	**execute**	Compromise Forward	5	6	1382	45000
8	Intruder	choose	Compromise Backward	2	5	-	-
8	Admin	**execute**	Patch	5	$\perp$	52	11000
12	Intruder	`fail`	Compromise Backward	2	5	1382	50000
12	Intruder	choose	Compromise Backward	4	6	-	-
16	Intruder	**execute**	Compromise Backward	4	6	1403	55000
16	Intruder	choose	Compromise Backward	1	4	-	-
20	Intruder	**execute**	Compromise Backward	1	4	1404	60000
20	Intruder	choose	Compromise Backward	2	4	-	-
24	Intruder	**execute**	Compromise Backward	2	4	1405	65000
24	Intruder	choose	Compromise Forward	2	5	-	-
28	Intruder	**execute**	Compromise Forward	2	5	1436	70000
28	Intruder	choose	Compromise Backward	3	5	-	-
32	Intruder	**execute**	Compromise Backward	3	5	1437	75000

Fig. 2. Play example Intruder maximum payoff

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

Vertices values are used to express which services/files 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 approaches 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 [29] 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. It is of course possible to use other means to compute vertex and dependency values but this is out of the scope of this paper. A sketch of this `vertexValue` algorithm is :

```
1)vertexValue(vertex) {
2) value = 1
3) foreach pred do
4) value += vertexValue(pred) x dependencyValue(node, pred)
5)  end foreach
6) return value
7)}
```

At line 2 the value of the node is set to 1. This is the base value for every vertex of the graph. This ensures that even a client (a vertex without predecessor) has a non-zero value. The algorithm is called recursively at line 4 in order to compute each predecessor value (Note that in case of circular dependencies the vertex own value needs to be excluded and we use a arbitrary order). Each predecessor value is weighted by the dependency value. The function `dependencyValue()` is used to compute the dependency value using the following formula:

$$dependencyValue = (occur/totalOccur) \times 100$$

where `occur` is the number of occurrences observed for a given dependency and `totalOccur` is the total number of occurrences observed for all the dependencies. Applied to our DG example (figure 1) with dependency frequency set by hand it gives the values detailed in figure 3.

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. 3. On the left dependency frequency values and on the right computed vertices values

4 Strategy

Intuitively a strategy is a succession of actions that a player can do to achieve his objective. For example the incident player may want a strategy to compromise network services. In this case the best strategy is the one that maximizes the number of services compromised. Similarly the administrator may want to have a strategy for patching her network that minimizes the patch cost and the time required and such that every vulnerable services are patched. The best strategy for a given set of objectives is found by looking at every plays generated by the game and keeping the one that maximizes/minimizes the most player objectives while satisfying his constraints. Constraints for instance can be that every hosts on the networks are not vulnerable at the end of the play. Formally a strategy is defined as:

Definition 1 (Strategy). *A strategy is the tuple* $S : (name, \mathsf{P}, \mathcal{O}, \mathcal{R}, \mathcal{C})$ *where* `name` *is the strategy name,* P *its owner,* $\mathcal{O}$ *is the strategy objectives set,* $\mathcal{R}$ *is the objectives priority strict order and,* $\mathcal{C}$ *is the set of constraints.*

In this paper, we consider memoryless strategies where rules are executed as fast as possible. This reflects that both players want to be the fastest possible and have no prior knowledge of how their opponent will react. This is the typical case when a new attack or intruder is faced. An other direction would have be to focus on iterated game where previous attempts are taken into account but since AG primarily aim at analyzing new situation and foreseeing the impact of network evolution, we choose to focus on memoryless strategy.
Iterated game can be used to describe the behavior of an attacker that use multiple attempts to refine is attack strategy against a network. Using such approach does not introduce a significant difficulty because an iterated strategy is an iterated memoryless strategy where some plays are removed at each iteration. Therefore by proving that finding a memoryless strategy is decidable, we also prove that finding an iterative strategy is decidable.

4.1 Objectives

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

Definition 2 (Play outcomes). *is the unordered set of natural numbers* $\phi : \{\mathtt{payoff}, \mathtt{cost}, \mathtt{opayoff}, \mathtt{ocost}, \mathtt{time}\}$ *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 strategy 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 objective that seeks to minimize the time and the cost is written

$$MIN(cost) \wedge MIN(Time).$$

Accordingly the order for the objective priority can be either

$$\mathcal{R} : cost > time \text{ or } \mathcal{R} : time > cost.$$

4.2 Constraints

Strategy constraints are used to consider only plays that are relevant to a strategy. In the patch strategy example, relevant 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. See 4 for a strategy in which at the end of the play no service is compromised and figure 4.3 for a strategy in which no service is ever compromised.

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. Thus the strategy set of constraints is expressed in the following fragment of CTL:

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

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	Intruder	choose	Compromise 0 Day	4	$\perp$	-	-
0	Admin	choose	Patch	4	$\perp$	-	-
2	Intruder	**execute**	Compromise 0 Day	4	$\perp$	21	20000
2	Intruder	choose	Compromise 0 Day	5	$\perp$	-	-
3	Admin	**execute**	Patch	4	$\perp$	21	500
3	Admin	choose	Patch	5	$\perp$	-	-
4	Intruder	**execute**	Compromise 0 Day	5	$\perp$	52	40000
4	Intruder	choose	Compromise Forward	5	6	-	-
6	Admin	**execute**	Patch	5	$\perp$	52	1000
8	Intruder	`fail`	Compromise Forward	5	6	52	45000

Fig. 4. A strategy where no service is compromised at the end of the play

the Boolean value † is true only on the last state of the play. This is a standard way to express a property that needs to be verified only at the end of play. In this fragment the patching strategy set of constraints 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:

$$\Box\neg Compr \wedge (\dagger \wedge\neg\, Vuln)$$

4.3 Dominant Strategy

The natural question that arises is what class of strategies should be considered for network security. A naive idea would be to consider the class of strategies 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:

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

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

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	Intruder	choose	Compromise Public	4	$\perp$	-	-
0	Admin	choose	Patch	5	$\perp$	-	-
3	Admin	**execute**	Patch	5	$\perp$	31	500
3	Admin	choose	Patch	4	$\perp$	-	-
6	Admin	**execute**	Patch	4	$\perp$	52	1000
7	Intruder	`fail`	Compromise Public	4	$\perp$	0	5000

It is obvious that 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. It can be intuited as the best option 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 called strictly dominant strategies [33]. In our example such a strategy exists for the administrator (see below). This is not always the case for a given player, for instance if the administrator can't firewall her public services then the intruder has a dominant strategy that uses 0 day exploit (Appendix B) and the administrator has a weakly dominant strategy that is able to dominate every none 0 day strategy.

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	Intruder	choose	Compromise 0 Day	4	$\perp$	-	-
0	Admin	choose	Firewall	4	$\perp$	-	-
1	Admin	**execute**	Firewall	4	$\perp$	0	10000
1	Admin	choose	Firewall	5	$\perp$	-	-
2	Intruder	`fail`	Compromise 0 Day	4	$\perp$	0	20000
2	Intruder	choose	Compromise 0 Day	5	$\perp$	-	-
2	Intruder	choose	Compromise 0 Day	5	$\perp$	-	-
2	Admin	**execute**	Firewall	5	$\perp$	0	20000
2	Intruder	choose	Compromise 0 Day	5	$\perp$	-	-
2	Admin	choose	Patch	4	$\perp$	-	-
4	Intruder	`fail`	Compromise 0 Day	5	$\perp$	0	40000
5	Admin	**execute**	Patch	4	$\perp$	21	20500
5	Admin	choose	UnFirewall	4	$\perp$	-	-
6	Admin	**execute**	UnFirewall	4	$\perp$	21	20500
6	Admin	choose	Patch	5	$\perp$	-	-
9	Admin	**execute**	Patch	5	$\perp$	52	21000
9	Admin	choose	UnFirewall	5	$\perp$	-	-
10	Admin	**execute**	UnFirewall	5	$\perp$	52	21000

4.4 Time Complexity

We now study the decidability of finding a strategy in AG. The key issue is that plays can be infinite. However they are ultimately periodic paths and therefore even when a game has an infinite play it is possible to decide which play is suitable for a given strategy. To decide if a play is suitable for a given strategy two things must be known: the play outcome and if the play satisfies strategy constraints. For strategy outcome we have the following result:

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

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.

For constraint satisfiability we have the following result:

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

Proof. As explained in proof 4.4, 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 the strategy constraint 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.

This lead us to the central theorem for decidability:

Theorem 1. *Strategies are decidable for any play by looking at a finite number of states.*

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

Moreover we can prove that deciding if a play satisfies a strategy can be done in polynomial:

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

Proof. By theorem 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 [23] holds. This theorem states that path model checking for CTL can be done in PTIME if the number of states is finite.

Ultimately we have the general decidability theorem:

Theorem 3 (Decidability). *Finding a finite number of strategies in an anticipation game is decidable.*

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 finite number of strategies in AG is decidable.

4.5 Memory Space complexity

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

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.*

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.

5 Evaluation

The evaluations were conducted on a 2.93 Ghz Intel core 2 Linux by running each analysis 3 times and using the mean. Two type of evaluation were performed: memory usage and speed impact. To have a more measurable example,

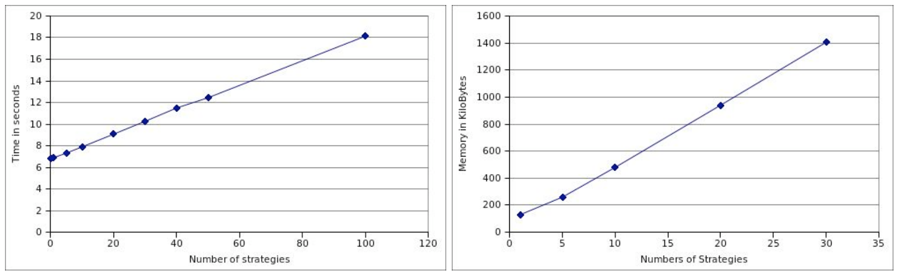


Fig. 5. Strategies impact on analyzer Speed and Memory

we added 3 more clients to the AG presented in this paper. Analyzing this AG with no strategy require 6.8 seconds.

First we measured the performance impact of strategies on NetQi speed. To do so we have run the analyzer with an increasing number of strategy requests. Execution time is the real time reported by the standard **time** command. 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 are reported in diagram 5. It is visible that the time requested to analyze the game grows linearly in the number of strategies. This is consistent with the theorem 2 because strategy validity is only evaluated on plays that give a better outcome. This also shows that the impact of finding a reasonable number of strategy is negligible.

Secondly we use the memory profiler **massif** from the **valgrind** tool suite [25] to verify that the memory needed by the analyzer grows linearly in the number of strategies as proved in theorem 4. We run test with a growing number of strategies ranging from 1 to 30. For each test we take the memory peak reported in massif diagram. Results displayed in 5 show that as expected the memory needed is linear in the number of strategies.

6 Conclusion

In this paper we have introduced memoryless strategies for anticipation game. We have detailed how they allow to take into account the financial aspect of the network security. We have proved that they are decidable and that they required only a linear space. We have also proved that verifying the validity of a strategy against a given play can be done in polynomial time. We have provided an algorithm to compute automatically network node values and finally we have verified the usability of strategies for practical use by fully implementing AG with strategy in a prototype written in C. Future work involves extending strategies with non-determinism to model attacker 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.
2. R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
3. M. Artz. *NetSPA : a Network Security Planning Architecture*. PhD thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science., 2002.
4. M. R. B. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.
5. J. Balthrop, S. Forrest, M. E. J. Newman, and M. M. Williamson. Technological networks and the spread of computer viruses. *Science*, 304:527–529, Apr 2004.
6. 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.
7. E. Bursztein. Netqi http://www.netqi.org.
8. 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.
9. A. Church. Logic, arithmetics and automata. In *Congress of Mathematician*, pages 23–35. Institut Mittag-Leffler, 1962.
10. V. Colizza, A.Barrat, M. Barthelemy, and A. Vespignani. The modeling of global epidemics: stochastic dynamics and predictability. *Bulletin of Mathematical Biology*, 68:1893–1921, 2006.
11. F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *Symposium on Research in Security and Privacy*, pages 202–216. IEEE Computer Society, 2002.
12. F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*, pages 197–216, London, UK, 2000. Springer-Verlag.
13. M. Dacier, Y. Deswarte, and M. Kaaniche. Models and tools for quantitative assessment of operational security. In *12th International Information Security Conference*, pages 177–186, May 1996.
14. L. de Alfaro, M. Faella, T. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *14th International Conference on Concurrency Theory*, volume 2761 of *LNCS*, pages 144–158. Springer-Verlag, 2003.
15. T. Henzinger and V. Prabhu. Timed alternating-time temporal logic. In *Formats 06*, volume 4202, pages 1–18. Springer-Verlag, 2006.
16. K. Ingols, R. Lippmann, and K. Piwowarski. Practical attack graph generation for network defense. In *ACSAC '06: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*, pages 121–130, Washington, DC, USA, 2006. IEEE Computer Society.
17. S. Jha, O. Sheyner, and J. Wing. Two formal analysis of attack graphs. In *CSFW '02: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, pages 49–63, Washington, DC, USA, 2002. IEEE Computer Society.
18. F. Laroussinie, N. Markey, and G. Oreiby. Model-checking timed atl for durationnal concurrent game structures. In *FORMATS 06*, volume 4202 of *LNCS*, pages 245–259. Springer-Verlag, 2006.
19. R. Lippmann, S. Webster, and D. Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *RAID '02: Proceedings of the 5th International Workshop on Recent Advances in Intrusion Detection*, pages 307–326. Springer-Verlag, Oct 2002.

20. K.-w. Lye and J. M. Wing. Game strategies in network security. *Int. J. Inf. Sec.*, 4(1-2):71–86, 2005.
21. A. Mahimkar and V. Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *18th IEEE Computer Security Foundations Workshop (CSFW), Aix-en-Provence, France, June 2005, pp. 287-301. IEEE Computer Society, 2005.*, pages 287–301. IEEE Computer Society, Jun 2005.
22. O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems (extended abstract). In *STACS 95*, pages 229–242, 1995.
23. 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. 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.
25. N. Nethercote, R. Walsh, and J. Fitzhardinge. Building Workload Characterization Tools with Valgrind. *Workload Characterization, 2006 IEEE International Symposium on*, pages 2–2, 2006.
26. 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.
27. 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.
28. X. Ou, W. F. Boyer, and M. A. McQueen. A scalable approach to attack graph generation. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 336–345, New York, NY, USA, 2006. ACM Press.
29. 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.
30. 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.
31. 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.
32. C. Ramakrishan and R. Sekar. Model-based analysis of configuration vulnerabilities. In *Journal of Computer Security*, volume 1, pages 198–209, 2002.
33. E. Rasmusen. *Games and Information*. Blackwell publishing, 2007.
34. 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.
35. 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.
36. B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's journal*, Dec. 1999.
37. B. Schneier. *Secrets & Lies: Digital Security in a Networked World.* Wiley, 2000.
38. 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.

39. H. R. Shahriari and R. Jalili. Modeling and analyzing network vulnerabilities via a logic-based approach.
40. 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.
41. L. P. Swiler. A graph-based network-vulnerability analysis system. In *New Security Paradigms Workshop*, pages 71 – 79. ACM Press, 1998.
42. S. J. Templeton and K. Levitt. A requires/provides model for computer attacks. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 31–38, New York, NY, USA, 2000. ACM Press.
43. 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.
44. 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.
45. 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.

A Administrator Weakly Dominant Strategy

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	Intruder	choose	Compromise Public	5	$\perp$	-	-
0	Admin	choose	Patch	5	$\perp$	-	-
4	Admin	**execute**	Patch	5	$\perp$	31	500
4	Admin	choose	Patch	4	$\perp$	-	-
7	Intruder	`fail`	Compromise Public	5	$\perp$	0	5000
7	Intruder	choose	Compromise 0 Day	4	$\perp$	-	-
8	Admin	**execute**	Patch	4	$\perp$	52	1000
10	Intruder	`fail`	Compromise 0 Day	4	$\perp$	0	25000

Fig. 6. Administrator weakly dominant strategy

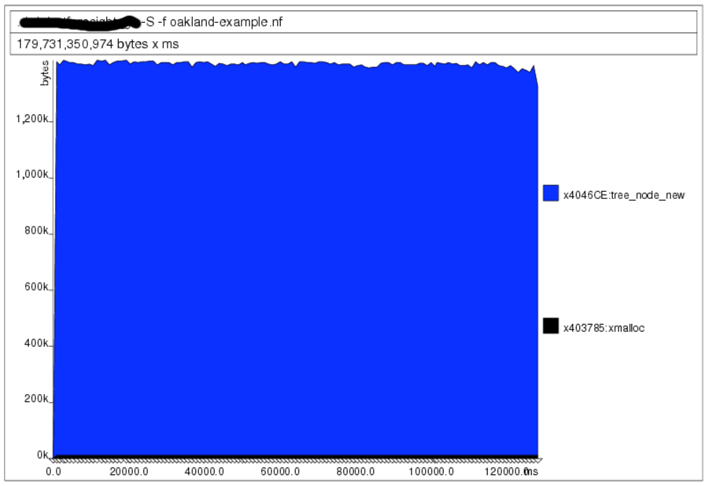
B Intruder Dominant Strategy

Time	Player	Action	Rule	Target	Succ	Payoff	Cost
0	Intruder	choose	Compromise 0 Day	4	$\perp$	-	-
0	Admin	choose	Patch	5	$\perp$	-	-
2	Intruder	**execute**	Compromise 0 Day	4	$\perp$	21	20000
2	Intruder	choose	Compromise Forward	4	6	-	-
3	Admin	**execute**	Patch	5	$\perp$	31	500
3	Admin	choose	Patch	4	$\perp$	-	-
6	Intruder	**execute**	Compromise Forward	4	6	1372	25000
6	Intruder	choose	Compromise Backward	1	4	-	-
6	Intruder	choose	Compromise Backward	1	4	-	-
6	Admin	**execute**	Patch	4	$\perp$	52	1000
10	Intruder	`fail`	Compromise Backward	1	4	1372	30000
10	Intruder	choose	Compromise Backward	4	6	-	-
14	Intruder	**execute**	Compromise Backward	4	6	1393	35000
14	Intruder	choose	Compromise Backward	1	4	-	-
18	Intruder	**execute**	Compromise Backward	1	4	1394	40000
18	Intruder	choose	Compromise Backward	2	4	-	-
22	Intruder	**execute**	Compromise Backward	2	4	1395	45000
22	Intruder	choose	Compromise Forward	2	5	-	-
26	Intruder	**execute**	Compromise Forward	2	5	1426	50000
26	Intruder	choose	Compromise Backward	3	5	-	-
30	Intruder	**execute**	Compromise Backward	3	5	1427	55000

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

C Analyzer Memory Consumption

The memory required by the analyzer is almost constant during the execution as shown in the diagram 8. This is partially explained by the fact that optimal strategies tends to be found during in the first plays because they are those who use the fastest rules possible.



```
[redacted]-S -f oakland-example.nf
179,731,350,974 bytes x ms
```

Fig. 8. Analyzer memory consumption for the execution with 30 strategies reported by Massif the Valgrind tool