Automating Cyber Defence Responses using Attack-Defence Trees and Game Theory

Ravi Jhawar[1], Sjouke Mauw[1], Irfan Zakiuddin[2]
Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg[1]
Noumena Research Ventures Ltd., Croydon, UK[2]
ravi.jhawar@uni.lu
sjouke.mauw@uni.lu
noumena.research.ventures@gmail.com

Abstract: Cyber systems that serve government and military organizations must cope with unique threats and powerful adversaries. In this context, one must assume that attackers are continuously engaged in offence and an attack can potentially escalate in a compromised system. This paper proposes an approach to generate defensive responses against on-going attacks. We use Attack-Defence Trees (ADTrees) to represent situational information including the state of the system, potential attacks and defences, and the interdependencies between them. Currently, ADTrees do not support automated response generation. To this end, we develop a game-theoretic approach to calculate defensive responses and implement our approach using the Game Theory Explorer (GTE). In our games, Attackers and Defenders are the players, the pay-offs model the benefit to each player for a given course of action, and the game's equilibria is the optimal course of action for each player. Finally, given the dynamic nature of cyber systems, we keep our ADTrees and the corresponding game trees up-to-date following the well-known OODA (observe, orient, decide, act) loop methodology.

Keywords: Cyber Defences, Attack Modelling, Game Theory, Security, Incident Response

1. Introduction

Cyber systems are becoming highly complex with ever-increasing dependencies both internally as well as with strategic partners and commercial service providers. Military organizations and critical businesses are also relying heavily on such cyber systems to meet their operational demands and to support mission execution. At the same time, cyber attacks are becoming stealthy and sophisticated, posing potentially very high damaging impact. In this context, a holistic framework for responding to cyber attacks becomes essential and it must encompass several functions including:

i) efficient collection of cyber situational information,
ii) analysis of possible attacks,
iii) determining the courses of actions in response, and
iv) taking the appropriate actions.

This paper focuses mainly on the 'determining the courses of actions in response' component of a deployed cyber system. We assume that situational information including the system state and parameters, and attack and defence related information is available. In this work, we systematically represent the situational information using Attack-Defence Trees (ADTrees) (Kordy, Mauw, & Radomirovic, Attack-defence trees, 2014). ADTrees improve the widely used attack trees formalism, by including not only the actions of an attacker, but also possible counteractions of a defender. The root node in an ADTree represents the attacker's (or defender's) goal and the children of a given node represents its refinement into sub-goals. Each node can have one child of the opposite type, representing the node's counteraction, which can be refined and countered again. The leaves of an ADTree represent the basic actions of an agent, which need not be refined any further.

Formally, ADTrees extend the formalism of defence trees (Bistarelli, Fioravanti, & Peretti, 2006), where defensive measures are not refined and can only be attached to leaf nodes. ADTrees can also be seen as merging attack trees and protection trees (Edge, Dalton, Raines, & Mills, 2006) into one formalism. Protection trees are AND-OR trees depicting how defensive measures can be refined into simple actions. Given the high expressivity and intuitiveness of ADTrees, complemented with strong mathematical foundations, they seem as an appropriate choice to describe and analyze cyber situational information.

Currently, ADTrees are used to analyse and quantitatively assess security scenarios. They do not compute the course of actions as responses against on-going attacks. We propose to address this limitation by applying game theory to ADTrees. Game theory provides a rich resource of mathematical and algorithmic tools to study the problems of competition or conflict. We view a *cyber response problem* as a game between an attacker who is competing to inflict some form of attack and a defender who is attempting to prevent the attack. A game-solver then computes the best responses to defend the cyber system from various attacks launched by an attacker.

Kordy et al. in (Kordy, Mauw, Melissen, & Schweitzer, 2010) have already established a two-way mapping and equivalence between games and ADTrees. However, they consider games of a highly restricted form that are not suitable in our context because of the following limitations:

- They use only binary pay-offs; this implies that there are only two possible outcomes: the attacker wins and the defender loses, and vice versa.
- They assume existence of perfect information implying that both players have full knowledge of all opponent's actions.
- Strict alternation of player's moves is required. Assuming an alteration between the attacker's and the defender's moves may be unrealistic in our case.
- Finally, the mappings in (Kordy, Mauw, Melissen, & Schweitzer, 2010) result in an increased abstraction from reality. Each mapping consists in generating a suitable syntactic object called an ADTerm that maps the binary pay-offs of the game tree. The way in which such syntactic objects represent the real world is unclear.

In this paper, we address the limitations in (Kordy, Mauw, Melissen, & Schweitzer, 2010) and define a game model that has the capability to represent the cyber response problem. Section 2 presents a motivating scenario that places our work in context. Section 3 provides the fundamental definitions of our game model and Section 4 defines the mapping between ADTrees and the basic form of our game model. Section 5 then extends our basic game to allow modelling of complex cyber response problems. Section 6 defines our approach for updating game trees following the OODA loop and Section 7 outlines our conclusions.

2. Motivating Scenario

Consider a military organization that has deployed a small, dedicated cyber system to support one of its missions. The mission might be for a Remotely Piloted Aircraft System (RPAS) to track an object in a geographical region. The cyber system performs functions like storing and processing the image and location data sent by the RPAS in order to generate the navigation plans (NP). This cyber system can also be a subnet separated by a firewall within a large distributed network operated by the military organization.

Assume that the dedicated cyber system consists of a file server (FS) which stores the image and location data, a navigation plan generator (NPG) that computes the future navigation routes for the RPAS, and three client workstations (WS) that control the RPAS. FS offers file transfer (ftp), remote shell (rsh) and secure shell (ssh) services to WS so that they can access the image and location data. NPG on the other hand allows WS and FS to execute commands on it using the ssh service. A firewall, which is intended to protect FS and NPG, only allows ftp, rsh and ssh traffic from WS to FS and NPG and blocks all other traffic. Let us further assume that there are vulnerabilities in ftp and ssh daemons, in the task scheduler of NPG, and in the address space resolution of FS's operating system. The access control list defines that a user has read and execute privileges while a root can read, write and execute. Finally, the goal of the attacker is to breach the integrity of the system so that the mission fails.

In ADTrees, attacks are represented as circles and defences as rectangles. Refinements are indicated by solid edges between nodes and counteractions are indicated by dotted edges. Attacks and defences can be refined conjunctively and disjunctively. A conjunctive refinement of a node has an arc connecting the edges going from this node to its children. A disjunctive refinement has simple edges.

The ADTree in Figure 1 shows how an attacker can modify critical mission data in two different ways and provides possible defence choices. In the first attack (see node "NPG root"), the attacker can obtain root privileges on NPG: first by gaining root privileges on WS via a key logging technique or by performing a buffer overflow attack on the ssh daemon. Then the attacker can use the interactive "cmd.exe" command. The defender can disable the task scheduler to prevent the execution of the cmd.exe command; use a two-factor authentication scheme against the key logging attack; and stop the ssh service to prevent buffer overflow.

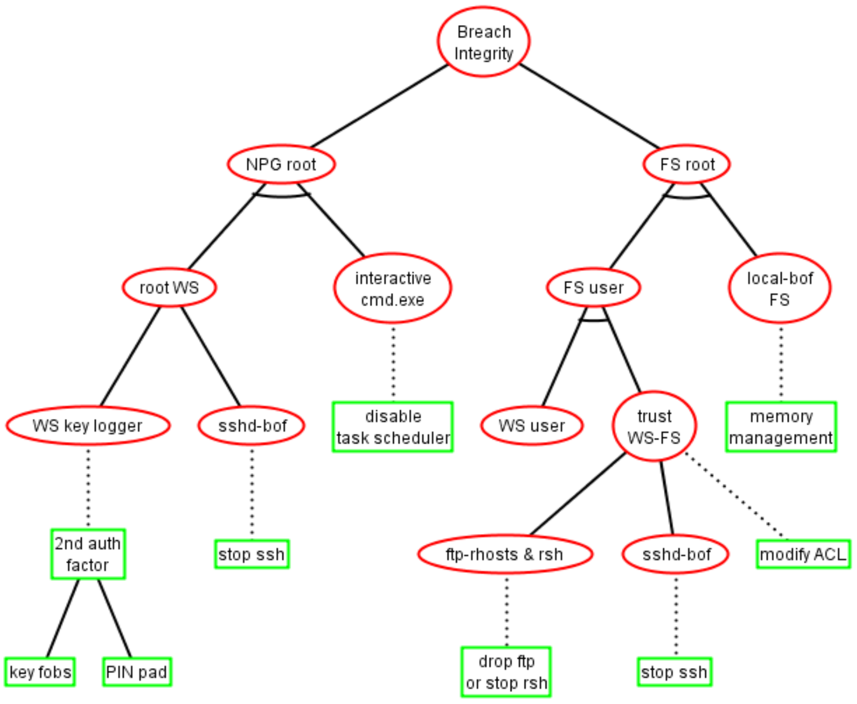


Figure 1 Example ADTree representing the attack-defence scenario for a military organization

In the second case, the attacker must obtain root privileges on FS (see node "FS root"). To achieve this, she must first gain user privileges on FS and then perform a local buffer overflow attack. The defender can prevent the latter attack by using adaptive memory management techniques. To obtain user privileges on FS, the attacker can, starting as a user on WS:

i) exploit the ftp vulnerability and use the rsh service to establish trust between WS and FS, or
ii) perform a buffer overflow using the vulnerability in the ssh daemon.

The defender can prevent this attack by:

i) modifying the access control list, or
ii) configuring the firewall to drop ftp packets from WS and blocking the rsh service, and
iii) stopping the ssh service.

3. Game Model

We use ADTrees to express situational information and to analyze attacks in the system. However, we note that ADTrees are only a language to describe and formalize attacks and defences; they do not compute responses against attacks by themselves. We propose to solve the cyber response problem by applying game theory on ADTrees. We define a game between an Attacker who is competing to inflict some form of attack and a Defender who is attempting to prevent the attack. A game-solver then provides the cyber responses to defend the system from the attacker.

In this section, we define the following basic components of our games: the game's players, knowledge states of the players, game's moves, and the pay-off function. The next section discusses an approach to generate our games from ADTrees.

3.1 Game's Players

Our model considers interaction between two players: a Defender and an Attacker. In general, considering a single Defender-player implicitly assumes that the Defender has nearly full knowledge of the state of the system and that she can implement any determined course of actions effectively. However, in comparison to centralized control, a model with localized decision-making seems sensible. To this end, one approach consists in defining multi-player games with a number of Defender-players. Another approach consists in coordinating several two-player games of a Defender against the Attacker, where each game makes a localized decision, and the overall solution is the composition of local results. We adopt the latter approach by defining two-player local games focusing on specific locations or critical resources in the system (e.g., a local game where the Attacker attempts to gain root privileges on FS and the Defender aims to protect FS). We choose the latter approach because it allows us to define game models for each resource in the system and to take into account the distinct trust levels associated with each resource. For example, a cross-boundary located server has lower trust than an on-site server. Our game model addresses this aspect by adjusting the pay-off values based on the `risk appetite' parameter.

The Attacker attempts to breach the defences of the system in order to disrupt missions. In our example, the goal of the Attacker is to breach the integrity of a mission either by compromising FS or NPG. Assuming a single attacker implies that she has full knowledge of possible attack strategies and has centralized control for inflicting her actions on the system. Therefore, having a single Attacker-player provides a model with a very strong attacker and it may be desirable to retain this model, irrespective of the aforementioned models for the defender.

3.2 Game's Moves and Knowledge States of the Players

When considering the game moves, we highlight a conceptual distinction between games and ADTrees, specifically regarding the notion of *strategy*. In a game, a strategy is a complete algorithm that tells a player what to do for every possible situation throughout the game. In ADTrees, concrete actions are only at the leaves and all other nodes define a refinement relationship using conjunction and disjunctive operators. Therefore, in an ADTree, the strategy is the connection between a concrete action and the goal that drives it.

For example, for the ADTree in Figure 1, an attacker can reach her goal following four *attack strategies*:

- a_1={WS user, ftp-rhosts & rsh, local-bof FS}
- a_2={WS user, sshd-bof, local-bof FS}
- a_3={WS key logger, interactive cmd.exe}
- a_4={ sshd-bof, interactive cmd.exe}

All actions within an attack strategy must be implemented to breach integrity, but implementing one of the four attack strategies is sufficient. While a_1 and a_2 allows the attacker to gain root privileges at FS, a_3 and a_4 compromises NPG. There are three *defence strategies* to protect FS:

- d_1={drop ftp or stop rsh, stop ssh}
- d_2={memory management}
- d_3={modify ACL}.

and two defence strategies to protect NPG:

- d_4={task scheduler} and
- d_5={2nd auth factor, stop ssh}.

Our goal here is to have the game's moves model the attack-defence strategies of each player.

The knowledge state of a player defines what the player knows of its moves and the moves of other players. The simplest case considers *perfect Information* where both players have full knowledge of all the moves of the game. In this work, we consider more complex models for knowledge states where the moves and pay-offs of both players are not fully known to either the Attacker or Defender. In particular, our game model allows moves where:

1. The Defender cannot distinguish the choices made by an Attacker and vice versa.
2. Game's moves are committed temporally independently and/or simultaneously.
3. A player may choose not to take any action that changes the state of the system (e.g., a defender can simply monitor the network for possible intrusions).
4. There is uncertainty in observations and expected pay-offs.

We believe that such complex game models can sufficiently represent the cyber security scenarios like the one described in Section 2.

3.3 Pay-off Function

Pay-offs model the benefit to each player for a given course of actions/game's moves. In zero-sum games, the gains of one player are equivalent to the losses of the other. We note that pay-offs are also critical in capturing essential notions like the `risk appetite'. Assigning realistic pay-offs is a hard problem and is out of the scope of the work presented here. Instead, we start from a basic game model with arbitrary pay-offs where the pay-off to the Attacker is simply a measure of the amount of work the Defender has to do. Therefore, when the Defender has to take one action the Attacker receives a pay-off of 1. Intuitively, the goal of each player is to commit their game moves such that they maximize their own pay-offs.

4. Mapping Attack-Defence Trees and Games

In this section, we define a basic game and provide its solution. We then describe the notion of equilibrium and its relationship with the cyber response problem.

4.1 The Basic Game Model and the Game Trees

To generate our basic game, we start from the ADTree that models the overall security of the system and compute all attack and defence strategies, as described in Section 3.2. In this section, we aim to model a two-player local game, focusing on a specific resource in the system (FS), as discussed in Section 3.1.

The players are clearly the Attacker and Defender; the game's moves that model the choices of each player are denoted as labels on the edges and take the form:

Player.InformationState.ActionType.

Player can take one of the two values *A* or *D* representing the Attacker and the Defender, respectively. *InformationState* is a number associated with each action representing the depth of the knowledge state of the player while making the game's move (see Section 3.2). In the simple example below, the decision for both players comes from their first information state. Finally, the *ActionType* refers to the concrete steps taken by each player – in our game, action types correspond to attack and defence strategies.

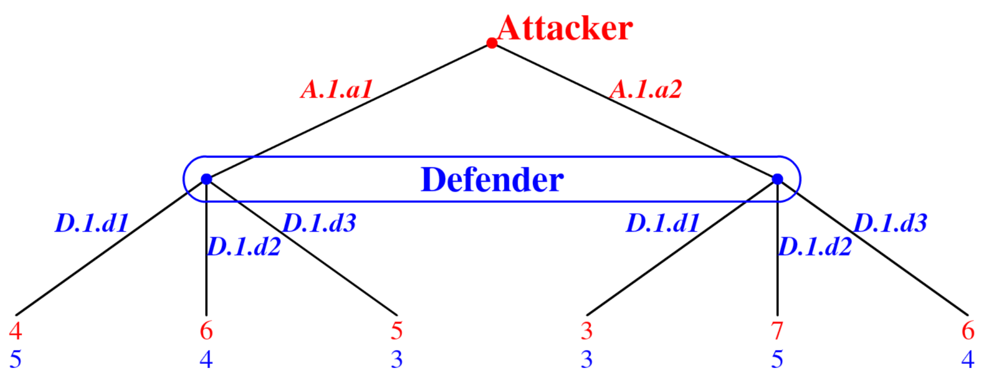


Figure 2 Basic game tree focussing on FS

For the local game focusing on FS, two attack strategies allow the Attacker to gain root privileges on FS:

- a_1={WS user, ftp-rhosts & rsh, local-bof FS}
- a_2={WS user, sshd-bof, local-bof FS}.

On the other hand, the Defender can protect FS – or respond to Attacker's moves – using one of the three defence strategies:

- d_1={drop ftp or stop rsh, stop ssh},
- d_2={memory management}, and
- d_3={modify ACL}.

For the sake of readability, action types in Figure 2 are denoted using the above attack and defence strategy identifiers. The Attacker must choose from a_1 and a_2 in order to implement its attack and, in response, the Defender must choose from d_1, d_2 or d_3.

A player's game strategy can be extracted by following a path from the apex to a leaf. The path presents the moves of both players. In Figure 2, the Defender circumscribes the two nodes that model the result of the Attacker's moves. We use this notation, following (Egesdal, et al., 2015) (Savani & Stengel, 2014), to denote the fact that the Defender cannot distinguish these nodes because it does not know the choice made by the Attacker. In other words, nodes circumscribed using a rounded rectangle are treated as a single information state for the Defender (see Section 5.2). Although the moves of the Defender are structurally presented as following the Attacker's moves, semantically there is no temporal dependency. This means that in our game model the moves are not assumed to be committed in any particular order and can even occur simultaneously.

The leaves of the game tree are the pay-offs and they measure the benefits to each player for their course of actions. For example, if the Defender chooses to strengthen the network by using memory management (strategy d_2) as a defence, although she can prevent the Attacker from gaining root privileges at FS, the number of tasks that the Defender performs is significant, thus the pay-off of 7 to the Attacker. We note that, although we have assigned the pay-offs here in a simple manner, we tried to take into account the risk appetite and the sensitivity of the impact of attack and defence choices.

4.2 Solving the Games

We use a web-based game solver called the Game Theory Explorer (GTE) (Egesdal, et al., 2015) to obtain solutions for our games. The solution consists in computing the equilibria, which in our case describes the best game strategies for both players. Attacker and Defender are in equilibrium if the Attacker is choosing the best strategy she can, taking into account the Defender's strategy, while the Defender's decision remains unchanged. Similarly, the Defender is choosing the best strategy she can, taking into account the Attacker's decision, while the Attacker's decision remains unchanged.

Figure 3 illustrates a part of the solution of our basic game (Figure 2) as provided by the GTE. The 2x3 matrices characterize the pay-offs of both players (player 1 being the Attacker and player 2 the Defender). Each of the three rows EE1, EE2 and EE3 denote an equilibrium, with corresponding expected pay-offs.

```
Strategic form:

2 x 3 Payoff player 1                         2 x 3 Payoff player 2

        D.1.d1  D.1.d2  D.1.d3                        D.1.d1  D.1.d2  D.1.d3
A.1.a1       4       6       5               A.1.a1       5       4       3
A.1.a2       3       7       6               A.1.a2       3       5       4

EE = Extreme Equilibrium, EP = Expected Payoffs

Rational:
EE 1   P1: (1) 2/3 1/3 EP= 5     P2: (1)  1/2 1/2 0 EP= 13/3
EE 2   P1: (2)   1   0 EP= 4     P2: (2)    1   0 0 EP=    5
EE 3   P1: (3)   0   1 EP= 7     P2: (3)    0   1 0 EP=    5
```

Figure 3 Solution of the basic game

Let us look at the equilibria in detail.

- (Row 1) EE1 consists of player 1 (the Attacker *A*) playing a game strategy labelled (1) and this strategy is for her to make the first game move A.1.a_1 with probability 2/3 and the second game move A.1.a_2 with probability 1/3. As a response, player 2 (Defender *D*) can make game moves D.1.d_1 and D.1.d_2 with equal probabilities of 1/2 each, but does not play D.1.d_3. By following this game strategy, the Attacker can expect a pay-off of 5 and the Defender's expected gains are 13/3.
- (Row 2) The equilibrium EE2 consists of a game strategy labelled (2) where the Attacker only makes its first game move A.1.a_1 with probability 1. As a response, the Defender also makes only the game

move D.1.d_1 with probability 1. The expected pay-offs through this game strategy is 4 for the Attacker and 5 for the Defender.

- (Row 3) The final equilibrium consists of game strategy (3) where the Attacker only makes its second game move A.1.a_2 with probability 1 and the Defender responds to it through the game move D.1.d_2 with probability 1. The expected pay-offs for the Attacker is 7 and for the Defender is 5.

We can use *the equilibria to identify the best defence responses against on-going attacks*. For example, in game strategy (2), when the Defender identifies that there is an *on-going attack* on FS following attack strategy a_1, the *best response* for the Defender is to drop ftp packets between WS and FS and to stop ssh service on FS (i.e., apply d_1). This allows the Defender not only to stop the on-going attack but also to strengthen her system without paying heavily in terms of the amount of Defender tasks required.

Our basic game model satisfies the 2nd and partially the 1st requirement listed in Section 3.2. However, to accommodate complex models – satisfying all four requirements – we need to extend our basic game so that our cyber response problem can be modelled holistically.

5. Extended Game Models

Our basic game model assumes a static scenario where the players consider all options upfront and make a strategy choice, which fixes a definite course of action for each player. The limitations of the basic model are:

- A player may choose not to take an action that changes the state of the system. For example, a Defender may only monitor the network to observe the situation and an Attacker may perform reconnaissance. Such *wait* conditions may be necessary, but were not included in the basic game model. To address this, we add a *wait* game move to the strategy sets of the players (see Figure 6).
- In networked systems, several unexpected system events and on-going attacks may go unnoticed. Since such observations are critical for successful execution of missions, we need to enhance our game trees and introduce probabilistic branching that takes into account the uncertainties about what has happened (see Section 5.1).
- In our basic game both players know the pay-offs to each other. This is an unrealistic assumption since it implies that each player knows the impact of a course of action on both itself and, critically, on its opponent. To address this limitation, we introduce randomization to capture variable pay-offs (see Section 5.2).

5.1 Randomization to Capture Uncertain Observation

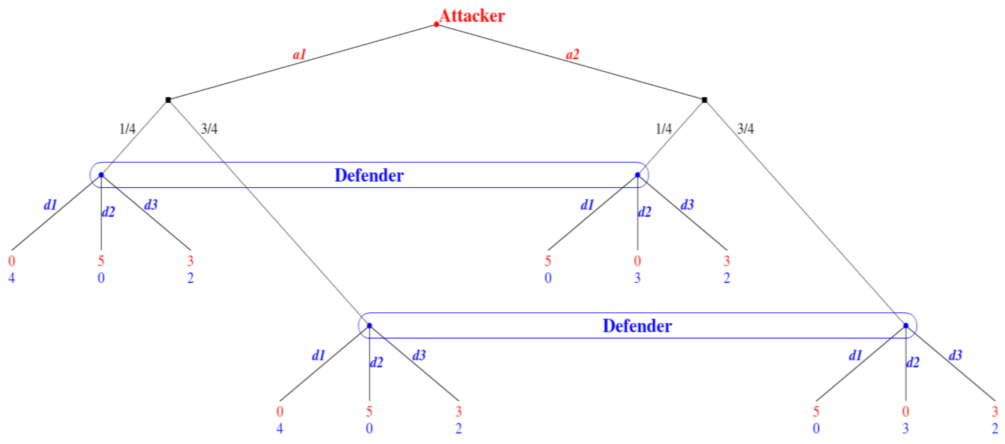


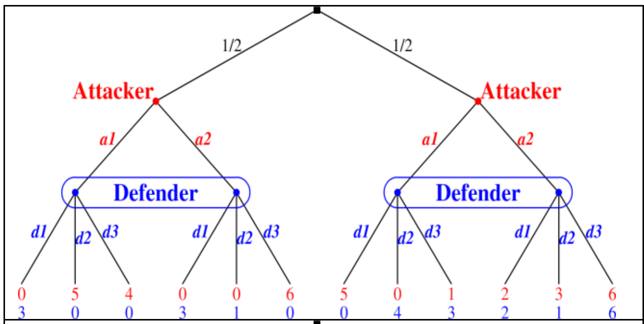
Figure 4 Game tree with uncertain observations

Figure 4 illustrates an example of how randomization in the game tree can model uncertainty of observation. The Attacker can choose either attack a1 or a2 (notation has been simplified here for readability). However, there is a probabilistic branch after her action, which leads to the choices for the Defender. Therefore, the Defender knows that there are 3/4 chances that the Attacker has committed a_1 and 1/4 chances of a_2 being played. In contrast, the Defender in our basic game did not know if Attacker plays a1 or a2.

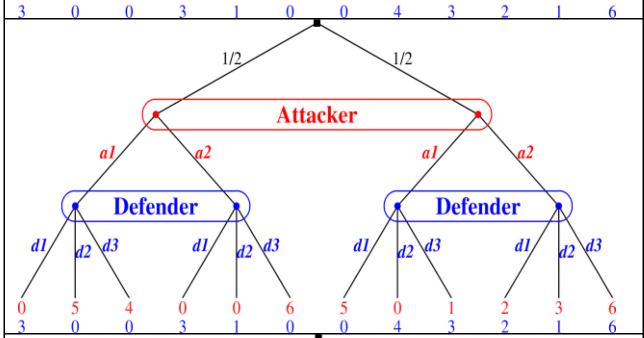
5.2 Randomization to Capture Variable Pay-offs

We consider four cases to introduce randomization of pay-offs. In the first game (Figure 5, row 1), there is an initial randomized branch with two sub-trees and corresponding pay-offs at the leaves. There are two possibilities of pay-offs and these are known to both players. As discussed in Section 4.1, observe that there is no rounded rectangle for the Attacker and two separate ones for the Defender, one for each case. In the second game (Figure 5, row 2), the Attacker does not know the pay-off possibility, but the Defender does, since we have an Attacker who is unable to distinguish the initial probabilistic branch. In the third game (Figure 5, row 3), neither the Attacker nor the Defender knows which pay-offs will be the case and in the final game, the Attacker knows which pay-off possibility will be the case, but the Defender does not.

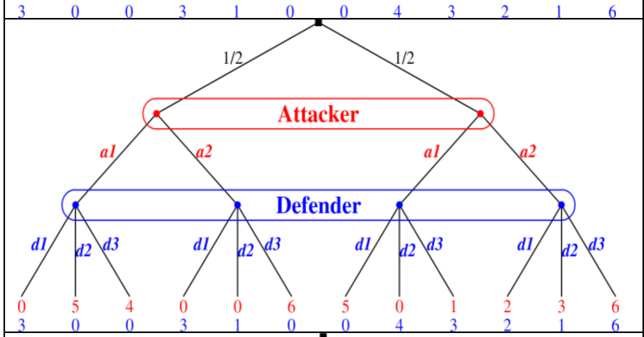
We note that, although the same initial pay-offs are assigned to all the game trees, the solutions are very different (see col. 2 in Figure 5). The number and the kind of game strategies in each equilibria and expected final pay-offs for both the players vary significantly since their knowledge states change drastically in each game. Following our extended game models, we can generate a rich set of game trees that precisely represent the complex requirements of our scenario.



```
EE 1
P1: (1) 0 1 0                        EP= 7/2
P2: (1) 0 0 1 0 0 0 0 0 0            EP=
15/2
```



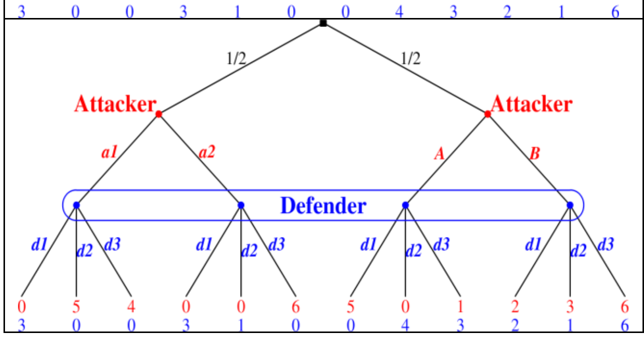
```
EE 1
P1: (1) 0 1                          EP= 3
P2: (1) 0 0 1 0 0 0 0 0 0            EP= 9/2
```



```
EE 1
P1: (1) 4/5 1/5                      EP= 5/2
P2: (1) 0 7/9 2/9                    EP= 9/5

EE 2
P1: (2)   0    1                     EP=   6
P2: (2) 0    0    1                  EP=   3

EE 3
P1: (3)   1    0                     EP= 5/2
P2: (3) 0    1    0                  EP=   2
```



```
EE 1
P1: (1) 0 0 0 1                      EP= 6
P2: (1) 0 0 1                        EP= 3
```

Figure 5 Game trees with randomization to capture variable pay-offs and corresponding game solution

6. Deploying Game Models with the OODA Loop

ADTrees capture cyber situational information in a static manner and support analysis of risks off-line. Cyber systems on the other hand are dynamic, with many system changes (e.g., migration of virtual machines, failure of storage disks) over time. Game trees are also a static formulation of interacting choices – a single tree cannot express the evolution of state over time. To address this issue, we propose to update our ADTrees and game trees in the events of system changes. We adopt the OODA loop methodology (Hightower, n.d.) as follows:

- *Observe* – collate information about cyber incidents and system changes.
- *Orient* – arrange collated information on suitable ADTrees.
- *Decide* – formulate games in concurrence with the updated ADTrees and solve them.
- *Act* – raise alerts to the system administrator with possible cyber response solutions to implement an appropriate action.

The loop reverts to the *Observe* step after *Act* and continues similarly thereafter. As an example, consider that at time instance *t*, it is *observed* that the Defender patches the ssh daemon and the Attacker is scanning for a new set of IP addresses in the network (*wait* game move). During *orientation*, attack a_1 is disabled and the corresponding defence d_1 need not "stop the ssh service" anymore (let d_1'={drop ftp or stop rsh}). Therefore, we update the game tree in Figure 2 and obtain the following game tree, which is then used to *Decide* about cyber responses.

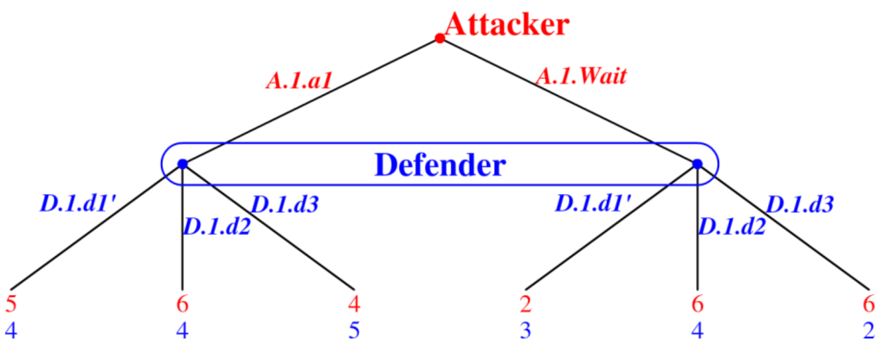


Figure 6 Updated game tree at time instance t

7. Conclusions

We proposed an approach to generate cyber defence responses by mapping situational information from ADTrees on to game trees. A variety of game models were demonstrated to support complex cyber response analysis, implemented by the GTE tool. Finally, we also account for dynamic system behavior by adapting our models following the OODA loop.

This work is an initial study that intends to understand the applicability of ADTrees and game theory in solving the problem of cyber responses generation. Our future work will first focus on defining a cost function that takes various functional and security parameters as input and provides the pay-off values as output. We will then focus on improving the scalability of our approach and perform experiments on realistic cyber testbeds.

Acknowledgements

The work of Sjouke Mauw and Ravi Jhawar was supported by the European Commission through the FP7 project TREsPASS (grant agreement n. 318003) and by the Fonds National de la Recherche Luxembourg through the ADT2P project (grant n. C13/IS/5809105).

Irfan Zakiuddin's contribution to this work was funded by the Defence Science and Technology Laboratory (DSTL), which is a part of the UK's Ministry of Defence. In DSTL, Irfan would like to thank Kevin Wise, the DSTL Technical Partner (TP), for another enjoyable project, much support and the usual string of great project meetings. Irfan would also like to thank Ed Moxon, the DSTL customer, for guidance and support. Irfan's work depended critically on huge amounts of free help from game theorists Rahul Savani and Theodore Turocy, both of whom repeatedly found time, amidst many more important commitments, to answer a barrage of questions.

8. References

Albanese, M., Jajodia, S., & Noel, S. (2012). Time-efficient and cost-effective network hardening using attack graphs. *DSN* (pp. 1--12). Boston, USA: IEEE.

Bistarelli, S., Fioravanti, F., & Peretti, P. (2006). Defence Trees for Economic Evaluation of Security Investments. *ARES* (pp. 416--423). Vienna, Austria: IEEE.

Edge, K., Dalton, G., Raines, R., & Mills, R. (2006). Using Attack and Protection Trees to Analyze Threats and Defences to Homeland Security. *MILCOM* (pp. 1--7). Washington, DC, USA: IEEE.

Egesdal, M., Gomez-Jordana, A., Pelissier, C., Prause, M., Savani, R., & Stengel, B. (2015). *Game Theory Explorer*. Retrieved from http://gte.csc.liv.ac.uk/gte/builder/

Hightower, T. (n.d.). *Boyd's OODA loop and how we use it*. Retrieved from Tactical Response: https://tacticalresponse.com/blogs/library/18649427-boyd-s-o-o-d-a-loop-and-how-we-use-it

Kordy, B., Mauw, S., & Radomirovic, S. (2014). Attack-defence trees. *Journal of Logic and Computation*, 24(1), pp. 55—87.

Kordy, B., Mauw, S., Melissen, M., & Schweitzer, P. (2010). Attack--Defence Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. *GameSec* (pp. 245--256). Springer.

Paul, S. (2014). Towards automating the construction & maintenance of attack trees: a feasibility study. *GraMSec* (pp. 31--46). Grenoble, France: EPTCS.

Savani, R., & Stengel, B. (2015). Game Theory Explorer: Software for the Applied Game Theorist. *Computational Management Science*, 12(1), pp.5—33.