A Game Theoretic Engine for Cyber Warfare

Allen Ott, Alex Moir and John T. Rickard

Abstract The nature of the cyber warfare environment creates a unique confluence of situational awareness, understanding of correlations between actions, and measurement of progress toward a set of goals. Traditional fusion methods leverage the physical properties of objects and actions about those objects. These physical properties in many cases simply do not apply to cyber network objects. As a result, systematic, attributable measurement and understanding of the cyber warfare environment requires a different approach. We describe the application of a mathematical search engine having inherent design features that include tolerance of missing or incomplete data, virtually connected action paths, highly dynamic tactics and procedures, and broad variations in temporal correlation. The ability efficiently to consider a breadth of possibilities, combined with a chiefly symbolic computation outcome, offers unique capabilities in the cyber domain.

1 Introduction

Game theory (1–15) is a mathematical theory of strategic behavior, in which a course of action (COA) consists of one or more individual moves taken by each player at a given stage of the game starting from their estimate of the current game state $S(k)$ at time k. A *game theory engine* is a computational device for advising a particular player as to the selection of future COAs based upon their estimate of the current state $S(k)$, given one or more evaluation functions $\varepsilon\,(S(j)),\ j = k+1, k+2,$ K

A. Ott (✉)
Distributed Infinity, Inc., 1382 Quartz Mountain Drive, Larkspur, CO 80118, USA
e-mail: aott@distributedinfinity.com

A. Moir (✉)
Distributed Infinity, Inc., 1230 N. Sweetzer Avenue, #302, Los Angeles, CA 90069, USA
e-mail: amoir@distributedinfinity.com

J. T. Rickard (✉)
Distributed Infinity, Inc., 4637 Shoshone Drive, Larkspur, CO 80118, USA
e-mail: terry.rickard@reagan.com

R.R. Yager et al. (eds.), *Intelligent Methods for Cyber Warfare*,
Studies in Computational Intelligence 563, DOI 10.1007/978-3-319-08624-8_10

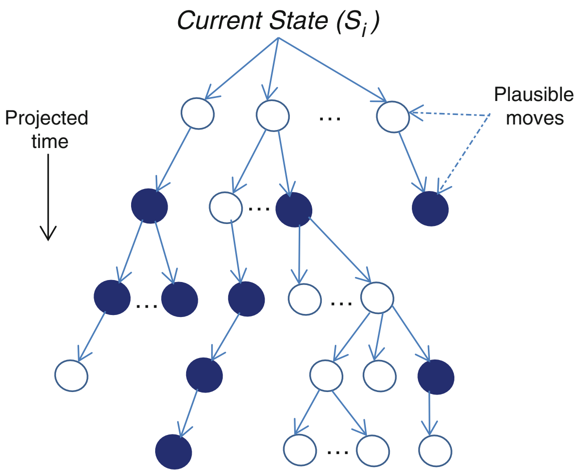


Fig. 1 Tree structure of a game theory engine output at a particular step of the game

that measure the change in the utility of future states that would result from a set of "moves" (i.e., actions taken) by himself and/or the other player(s).

The objective of a game theory engine is to identify for each player the set of feasible moves by all players from a given state, and to select the COA for a player that optimizes the sequence of future states with respect to his own assumptions at each step as the game proceeds. This optimal COA is also referred to as a "plan". The output of the game theory engine at each step of the game is represented as a tree with branches that are contingent upon the actions of all players. Figure 1 illustrates this structure in the case of a two-player game.

In general, each player in a game may have their own evaluation function, which in a two-player game we would denote by $\varepsilon_1 (S)$ and $\varepsilon_2 (S)$, respectively. In the simplest case, also known as a zero-sum game, the state S is commonly agreed by both players and assumed to represent the true state of the network, and the evaluation functions satisfy $\varepsilon_1 (S) = -\varepsilon_2 (S)$, i.e., one player's gain (loss) in value is equal in magnitude and opposite in sign to the other player's gain (loss).

In more complex and realistic cases, the current true state $S(k)$ of the network may not be available in full detail to one or more players. In such cases, each player may have their own unique (and perhaps only partially accurate) *estimates*, say $\hat{S}_1(k)$ and $\hat{S}_2(k)$, respectively, of the true state, while player 1 may have an estimate $\hat{S}_{12}(k)$ of the state perceived by player 2, and vice versa for player 2's estimate of the state $\hat{S}_{21}(k)$ perceived by player 1. Either player may assume the feasibility of certain moves by herself or her adversary that are in fact disallowed by the true state of the network. For example, a player may believe they know the password to a device and thus assume they can login and perform certain actions, when in fact they do not have the current password. The player may not be aware until a later time, if at all, that some of their moves were actually unsuccessful.

In addition to the potentially distinct estimates of state, the evaluation functions for each player may be different, so that for example in a two-player game, even if both players are in complete agreement on a common state S, the evaluation functions $\varepsilon_1(S)$, $\varepsilon_2(S)$, $\varepsilon_{12}(S)$ and $\varepsilon_{21}(S)$ may all be distinct. Thus, in addition to the true state, we may have to consider four different state estimates, as well as four different evaluation functions, at each step of a two-player game, and even more when additional players are involved.

Since a game theory engine is capable of considering a very large number of possible moves, it provides a natural mechanism for the modeling and analysis of cyber warfare offensive and defensive tactics. This is not to suggest that the human analyst can be replaced in this role. Instead, we consider this to be a useful tool to supplement the expertise of the human analyst by enabling the modeling of adversaries, the prediction of the efficacies of potential moves and the scoring of the vulnerability of a network (either one's own or an adversary's) to various cyber-attacks.

In this chapter,we describe a practical game theory engine denoted Themistocles that has been developed and employed over the past decade in cyber warfare analysis. At each time step k, Themistocles fuses past actions into a representation of the current state $S(k)$ of a network and the corresponding perceived states by all players. The latter states are functions of observed, inferred or hypothesized actions. Actions that could changethese state estimates may be observed directly (i.e., sensor data indicates the action), inferred indirectly from other observations, or hypothesized (neither observed nor implied by other objective evidence). All three types of actions involve varying degrees of uncertainty.

For each player, the feasible paths from their current state estimate and their estimate of their opponents' perceived state to a series of future states are constructed and scored with respect to the corresponding evaluation functions. The challenges are (1) to represent the evaluation functions of each player so that the scoring of each potential future state can be performed from the perspective of that player, and (2) to maintain a sufficient number of hypothesized future states to enable exploration of the full spectrum of path possibilities. In numerous formal cyber war games, Themistocles has demonstrated a capability to generate recommended COAs that met with the approval of human experts monitoring the games as being consistent with their best judgment of strategies and closely predicted the actions and tactics of human agents.

The remainder of this chapter is organized as follows. Section 2 describes the structure and algorithms of Themistocles. Section 3 presents examples of Themistocles' employment in cyber warfare scenarios. Section 4 concludes.

2 Themistocles Structure and Algorithms

Themistocles is comprised of four major software components, as illustrated in Fig. 2. The main processing flow is contained within the Search and Move Generation components. The Scheduler manages time within the game and initiates processing

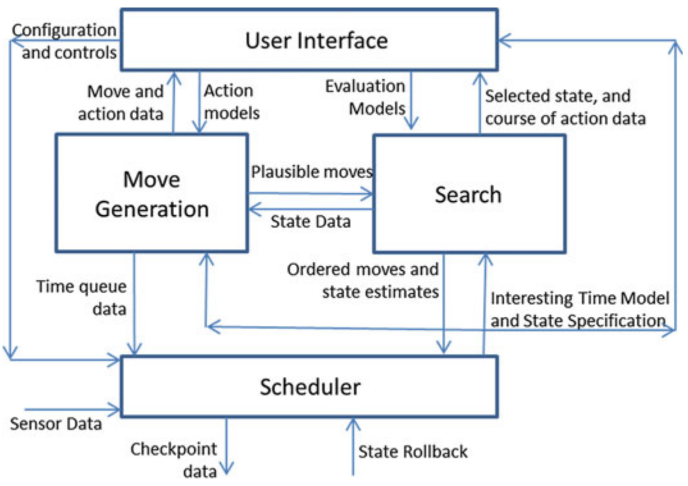


Fig. 2 Diagram of Themistocles components and their inputs/outputs

within the other components. The User Interface manages interaction with the user, including user controls and displays. This section first lays out the definitions required to specify system components and variables in Themistocles and then describes each of these in turn.

2.1 Definitions

The following definitions further detail the structure of Themistocles as a game-theoretic engine for modeling and analyzing cyber warfare scenarios.

Domain. The operating domain of Themistocles is a network of digital devices that are potentially capable of communicating with one another over this network. The devices may or may not be operable, and a given device may or may not be accessible by one or more players in a game.

State. The state in Themistocles is defined as the set of all variables and their associated values needed to characterize the system situation at a given point in time. In all cases, templates may be used to configure the system for ease of setting up a game. Global state variables characterize the overall situation on the network, while local state variables describe the situation on each device. Local state variables describe devices' pertinent attributes, operational status and accessibility. The number of local state variables is device specific, but may range upwards of 100 variables of different types. Examples of these include the on/off state of the device, whether it is a server, the operating system (O/S), the root password, as well as subjective variables such as the degree of belief in current suspicious activity on the device, etc. Global state variables include subjective variables such as the *work cost* of each prospective move,

Table 1 Partial list of local state variables for a network computer

Cyber object state variable	Value type
Admin password known	Boolean
BufferOverflow vulnerability	Boolean
Database service	Numerical
FTP service	Boolean
HTTP service	Boolean
Has account	Boolean
Has password file	Boolean
isCritical	Boolean
isHostKnown	Boolean
isHostUp	Boolean
isServer	Boolean
Patch version	Numerical
Physically accessible	Boolean
Physically at machine	Boolean
Root kit installed	Boolean
Needs investigation	Boolean
Key logger installed	Boolean
Suspicious activity seen	Numerical

Table 2 Partial list of global state variables for a network

Global state variable	Value type
Danger level	Numerical
Paranoia	Numerical
Political risk	Numerical
Risk	Numerical
Work	Numerical
Number of victims	Numerical
Number of decoys	Numerical

the *cumulative work cost* for each player, the *risk of discovery* for each prospective move for a player (usually only factored into the evaluation functions ofred players) and the cumulative *paranioa* of a blue player (used to determine the eligibility of certain drastic moves such as system restores).

Table 1 provides a listing of some typical local state variables for a network computer, while Table 2 provides a listing of some typical global state variables for the network. Each of these state variables has a set of sub-values corresponding to its true state and its estimated state by each player in a game, both for themselves and for the other players. Thus in a two-player game, there will be 5 sub-values for each state variable, i.e., truth state, player 1's own estimate of the state values and his estimate of player 2's assignment of state values, and similarly for player 2.

Table 3 Partial list of typical moves available to network defender/attacker

Defender move name	Move effect	Opponent observables
AnalyzeSystemLogs	Logs viewed, get info such as a login record or service installation	None
AnalyzeDataLogs	Database viewed, learn whether data was modified, deleted, or added	None
IP filter	Blocks a given set of IP addresses at the firewall	SYN flood stops working
InvestigateShutdown	Determine if a host shutdown was legitimate	None
NotifySecurity	Security team is on alert	More extreme counter moves
RestoreSystemFromBackup	Deletes all data and software, puts it back into standard start state	Lost connection, lost malicious services, lost backdoor
Attacker move name	**Move effect**	**Opponent observables**
Modify data	Corrupt data in a database	Tripwire alert on modified files
Port scan	Determine IP's of host's on a subnet and the services they offer	TrafficAnalyzer alert (i.e., Snort)
SetupBot	Take over a machine for later attacks	None
SQL injection	Gain root privileges by embedding string literal escape characters into the login command	None
SYN flood	Distributed denial of service (DDoS) by sending thousands of TCP SYN packets to a single machine	Service unusable or network slow

Move. A move is a member of a relatively small set of steps (i.e., O (10) actions) that a player can execute on a given device (e.g., a login). Moves have prerequisites (e.g., a device must be turned on, the player must know the login password, and the paranoia level makes the move eligible) and effects. The effects include a work cost associated with the move, which is expert-assigned, and changes in local or global state variable values (which in turn can add to the cumulative work cost for a player). Table 3 provides a listing of typical moves, along with their prerequisites and effects.
Move generation. Move generation is the process of creating a set of *feasible* moves for a particular player, given the current state of the system. This set also has relatively small cardinality in most instances, since a particular player typically has access to only a fraction of the devices on the network. As well, the values of both local and global state variables can further prune the set of feasible moves.

Objective function. An objective function is one of a set of utility functions for each player whose independent variables are local and/or global state variables of the system. These state variable values are mapped into negative or positive integer scores, with the sign depending upon whether a given state of the system confers negative or positive benefit to the player. The scores are expert-assigned on a relative scale on the interval $[-5 \times 10^4, 5 \times 10^4]$, and reflect the cost/benefit of system state variable values. Thus a score of 1000 (−1000) reflects 10 times the benefit (cost) of a score of 100 (−100). A weighted average of these scores is calculated over all of the objective functions for each player, and the result is mapped via a sigmoid function to the interval $[1, 10^5]$ to produce an overall score for a given system state, from that players' perspective. This range is chosen in order to provide adequate dynamic range to the normalization steps that map COA state scores back into *utility* values, the latter residing in the unit interval [0, 1]. Table 4 presents a typical set of objective functions for a blue player (defender), while Table 5 presents a typical set for a red player (attacker).

In addition to the state-related scores calculated from these objective functions, there are also work costs (with negative values) that reflect the time and expense associated with a given move. These costs are cumulated over sequences of moves, and the cumulative work cost upon arriving at a given state is deducted from the overall score associated with that state. In addition, there are scoring-related state variables such as the *danger level* for the network, a number in [0, 1] that determines how the scores of individual players' COAs are combined into a joint COA score, with a value of 0.5 giving equal weight, a value of 0 assigning all weight to the red players' score(s) and a value of 1 assigning all weight to the blue players' score(s).
Utility. Utility is a normalized score associated with each state involved in a particular COA, i.e., with each state resulting from a sequence of moves by the players in the game. The normalization is with respect to all feasible moves deriving from the current system state. Thus the utility of the successive states in a COA monotonically

Table 4 Blue player objective function descriptions

Objective function	Description
Preserve availability	Adds points for each host under supervision if the host is up and working properly
Investigate suspicious activity	Adds points for states that provide information about a host that has gone down or is non-functional, even if it isn't fixed
DoS defense	Adds points for maneuvers to stop a denial of service, such as blocking IP addresses, ports, or applying patches
Worm defense	Adds points for applying patches; deducts points for non-critical ports being open, deducts points for each host infected
Submit weekly report	Adds points for successfully uploading data to a database on a weekly basis
Minimize work	Deducts points for executing moves that utilize administrator time/energy

Table 5 Red player objective function descriptions

Objective	Description
Corrupt database	Adds points for modifying data on any database
Corrupt web server	Adds points for modifying data on any web server
Cover tracks	Adds points for removing log entries, software installations, etc. that result from an attack and could lead to being caught
DoS host	Adds points for preventing network access to any host
Gain server root account	Adds points for obtaining a username/password on a server
Minimize risk	Deducts points for executing moves that have risk
Poison DNS	Adds points for modifying host files to point to one of your own servers
Remote reconnaissance	Adds points for mapping an opponent's network and determining what services and vulnerabilities are there
Setup bots	Adds points for getting root privileges on remote machines
Steal data	Adds points for exfiltrating data from any host
Steal server data	Adds points for exfiltrating data from any server

decreases as the depth of the COA increases, and a COA is terminated when the utility of its leaf node falls below a cutoff threshold. This process is further described in the Search component below.

Action Queue. The Themistocles Action Queue manages the execution of all moves. It has two primary functions: (1) to test the effects of each move, and (2) to manage the time clock of the game. When a prospective feasible move is added to the action queue, the resulting state change is calculated and the game time is advanced to the next *interesting time*. The latter time is the minimum increment of time until the move generates an observable event, or until the move completes, or until a *pass time* is reached (i.e., the maximum increment of time permitted in the game scenario). Once a particular move has been added to the Action Queue, the overall utility of the resulting state is calculated and stored. Following this, the move is removed from the Action Queue and the next feasible move is added, with this process repeated until the respective utilities of all feasible moves have been calculated.

With these definitions, we now proceed to describe the four components of Themistocles in Fig. 2 and their interactions.

2.2 Search Component

The search engine is the core of the Themistocles software. The search engine performs the selection and evaluation of prospective feasible moves recursively over time. The output of the engine at each step in the game is data describing the prescribed COA and the states corresponding to each evaluated COA.

The Themistocles search process is a tree-based search designed quickly to produce an initial prescribed COA, using a leaf node score cutoff threshold in combination with a maximum tree depth, and then successively to refine the prescribed COA by deepening the search via iterative reductions in the cutoff threshold, as elaborated upon below. Each player performs his own tree search at each step of the game, based upon his own estimates of the system state and his estimates of the system state assumed by the adversary(s).

The root node of a player's search tree corresponds to the current state of the system $S(k)$ at time step k in the game. Starting with the root node, the tree is constructed in a partially serialized manner by considering the collection of child nodes $N^n_{i,j}$ at levels $n = 1, 2, K$ that would result from the feasible move sets of a given player and her adversary(s), where the superscript n refers to the depth of the tree, i indexes the child nodes at a given level n and j indexes the child nodes of parent node i ($n = 1$ corresponds to the children of the root node, for which only a single parent node exists, i.e., $i = 1$ only for this level). Each node in the tree corresponds to a prospective future system state $S^p_{ij}(k+n), n = 1, 2, K$ for player p, where these states are ordered in a time sequence dictated by the move effects.

A utility value $V\left(S^p_{ij}(k+n)\right)$ for player p is calculated for each child node, and these values are then normalized to sum to unity by dividing each child node value by the sum over all child node values, resulting in a normalized utility value $V'\left(S^p_{ij}(k+n)\right)$, i.e.,

$$V'\left(S^p_{ij}(k+n)\right) = \frac{V\left(S^p_{ij}(k+n)\right)}{\sum_j V\left(S^p_{ij}(k+n)\right)}. \tag{1}$$

For $n = 1$, i.e., the first level of child nodes from the root node, these values are equated to the corresponding nodes' game score denoted by $P^p_{ij}(k+1)$, i.e., $P^p_{1j}(k+1) \equiv V'\left(S^p_{1j}(k+1)\right)$. For $n > 1$ the $V'\left(S^p_{ij}(k+n)\right)$ are multiplied by the parent node's game score $P^p_{i'i}(k+n-1)$ (where i' is the index of the grandparent node), resulting in a game score for node $N^n_{i,j}$ given by

$$P^p_{ij}(k+n) = P^p_{i'i}(k+n-1)V'\left(S^p_{ij}(k+n)\right) \tag{2}$$

If $P^p_{ij}(k+n) > P_{cutoff}$ for a given child node $N^n_{i,j}$, then node $N^n_{i,j}$ becomes a parent node for a follow-on set of moves that deepens the tree. Thus the score of the terminal node for a given COA(path) through the tree decreases as the tree depth increases. Some moves may be generated with an associated probability, in which case the child node utility value prior to normalization is also multiplied by its (expert assigned) probability of occurrence.

The above process is continued until no parent node has a child node with a score above the cutoff threshold, at which point these parent nodes become the leaf nodes of the tree. This search for a particular player is illustrated in Fig. 3, where the white nodes represent the states resulting from that player's move choices and the dark

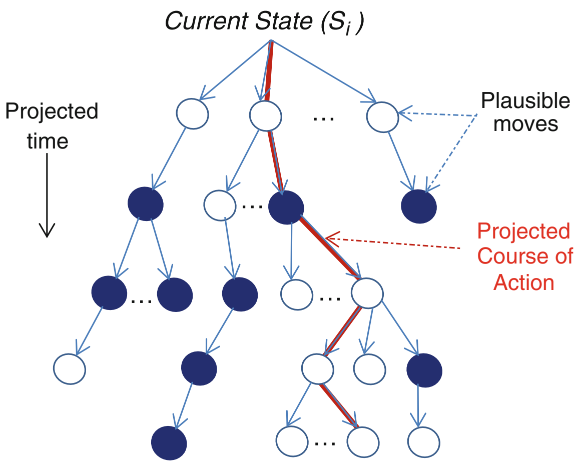


Fig. 3 Search tree from a given current state, showing projected moves by both players in a two-player game

nodes represent the states resulting from the adversary's move choices. Note that the search tree admits a combination of these states in any path through the tree.

The COA is scored based upon knowledge engineering defined abstract parameters specifying expected effects of moves and weights of objective functions for a given player evaluation model. Themistocles instantiates the moves and effects of each move and calculates the COA score at each node based upon the accrued move effects and weighted objective functions for each player. The path terminating at the highest scoring leaf node is selected as the prescribed COA and is then executed in an autonomous game simulation. In an interactive game, the top three scoring COAs are presented to the user, who then selects the one of his choice.

2.3 Move Generator

A move is defined to have the following characteristics:

- A list of preconditions.
- Effect on state upon initiation.
- Effect on state upon completion.
- A list of conditional effects during execution.
- Timing information for the entire move and each effect.
- A list of possible outcomes and probabilities for each effect.

In addition to explicitly modeling timing effects and stochastic move outcomes, the move set differs from traditional game move sets in another fundamental way. In a traditional game, two opposing players alternate moves. In most real-world domains such as cyber-warfare, this is simply not the case. Each player has the option of

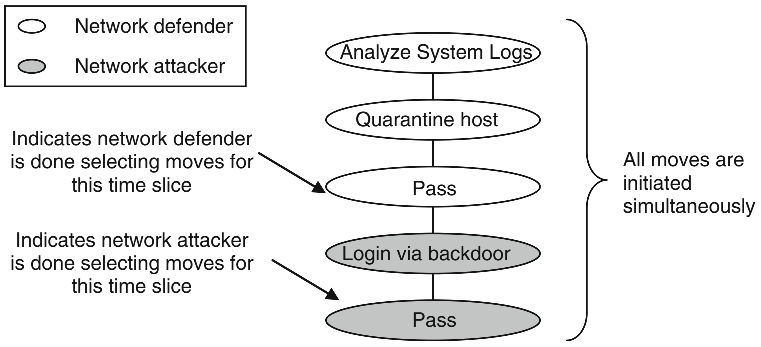

Network defender
Network attacker
Indicates network defender is done selecting moves for this time slice
Indicates network attacker is done selecting moves for this time slice
Analyze System Logs
Quarantine host
Pass
Login via backdoor
Pass
All moves are initiated simultaneously

Fig. 4 Snippet of partially-serialized game tree

choosing multiple moves that are executed simultaneously. In fact, both players will frequently be executing multiple actions at the same time.

To accommodate this, the search utilizes an untraditional approach to tree construction where all simultaneously chosen moves are serialized, such that they are listed in order in the tree despite the fact that they will be executed at the same time. This is accomplished by introducing a new move type: *Pass*. A pass indicates that the player will not be choosing to begin any additional actions until the next interesting time. All moves chosen before a pass are interpreted as beginning at the same simulated time. Figure 4 shows an example of a small subset of a partially serialized game tree.

When there are no players left to choose a move at a particular interesting time, the serialized move tree is parsed and the scheduler is notified of the postulated move selections for each player and the time queue entries for each action chosen.

While each move has a duration and set of possible outcomes associated with it, both players may or may not be aware of these outcomes. Awareness of the state of the network is based on available resources, and may be contingent on making moves to gain information. Even when a move produces an observable, such as a message logged by a deployed Intrusion Detection System (IDS), the players may not be in a position to see the observable without further action. In the event that a player can see the observable the system will give that player a chance to respond to the observed event.

Note that both players are not necessarily given the option of moving during a particular slice of time. Players only move if one of the defined events occurs such that they are aware of it. Thus, if the defender completes a move, the defender will have the option to choose more moves but the attacker will only have that option if the event has produced an observable to the attacker.

For added realism, there are three types of dynamic environmental moves included in any game: pure, scheduled, and consequential. Pure environmental moves include elements of the environment that have unknown or dynamically changing attributes. Schedule driven moves have scheduled preferential occurrence (such as circadian rhythm driven actions). Consequential environment effects simulate unexpected results tied to a specific state change or move execution. The actual state effects can be modeled that same way as player moves, using the action model. However, the triggering effect can be different.

Pure environmental actions are simulated using a Bernoulli or Poisson distribution tied to specific environmental factors. This technique was used effectively in training programs for noise generation. This simulation takes into account the state context so that the outcomes will make sense; for example, actions will not occur from an object that isn't capable of the proposed function.

For schedule driven factors, conditional effects are applied based upon the time of occurrence. The time simulation can be selected based upon the particular environmental factor being modeled. For example, this technique was used in space network operations to model environmental factors such as thunderstorms, which in some areas are highly dependent upon the time of day.

2.4 Scheduler

The Scheduler is responsible for maintaining the time queue of "interesting times". These interesting times are provided by move postulation, including start and end time, sensor data input, and user input.

When a move is executed, the scheduler inserts the selected moves into the action queue, advances time to the next interesting time, decides which player next has a turn, and calls move generation to provide that player with all available moves. The move start times are the times from the serialized move tree after composition of all projected player move selections. The move completion times are the times at which the last move effect is complete whether or not there is an observable state effect.

A move may have zero or more expected impacts to state and zero or more observables for any player as a result of the move execution at specified times during its execution. A Pass move has the simplest time data, having no effects on state times but a completion time. Any time a move completes, whether or not it had an impact on state, the Scheduler returns control to the move generator to determine whether or not another move of any type by any player should be initiated. Move generation augments the serialized move tree with moves selected for any player followed by a pass move.

The Scheduler maintains two interesting time queues. The primary simulated game time queue is the game action queue. The game action queue is maintained for all actions derived from human or computer selected moves, environment model moves, and sensor data reports. The secondary queue is the search action queue. This queue is maintained for all actions derived from projected moves by the search engine during a course of action evaluation. Any time an interesting time from the game action queue implies a change of state, the scheduler initiates a course of action evaluation by copying the game action queue to the search action queue, and initiating a search with the search action queue. The search action queue is then managed with projected moves and state changes selected by the search engine and driven by the search action queue.

When a course of action path has been evaluated to the point where the cutoff score has been reached, then the scheduler will back up the search action queue to the time of the last state that was not completely evaluated (paths with scores higher than the cutoff still exist), and reinitiate the search engine. When all paths have satisfied the cutoff, then the Scheduler will return to the game action queue and send results of the latest state search to the appropriate player. A computer (automated) player will always select the top scoring move(s), a human player can select moves indicated by the search results and/or any other set of valid moves. This process repeats continuously until there are no further actions in the game action queue. A condition where no further actions exist in the game action queue occurs when the game has been updated comprehensively but a human player has yet to finish move selection or when a predetermined time limit has been reached.

The scheduler may run Themistocles much faster than real time or much slower than real time. The scheduler runs Themistocles as fast as possible between the game action queue entries. If the primary (game action queue) processing is provided solely by computer input and not held for sensor data input, then Themistocles can run many times real time. If the cutoff score is set very low, requiring deeper and broader analysis (search action queue), or the moves in the primary (game action queue) are defined in very fine grained time or with operator delay, then Themistocles can run many times slower than real time.

The Scheduler also manages checkpoint retention, saving the state of the game at specified times. The checkpoint initiation can be based upon a change in state trigger, a simulation time trigger, or an operator trigger. As requested by the user, the Scheduler restores the game to a full fidelity image at a specified time from a stored checkpoint file.

2.5 User Interface

The User Interface (UI) leverages JAVA graphics packages to provide a graphical representation of the state. It displays sufficient information that the observer may understand the status of the game and the state of the target network. The UI may present multiple views from the perspective of individual players including state, resources, and moves available. In human mode, the player has an end turn button that basically completes the move selections for that interesting time and triggers the action queue to insert the pass move for the active display player. The UI also supports configuration of the game simulation including checkpoint and rollback.

The User interface connects to the game server using JAVA Remote Method Invocation (RMI) so that a client can be run on any other (potentially remote) machine. As well, any interface could be created and connected to the game server through the RMI.

3 Examples

This section presents an example of Themistocles' use in a realistic cyber warfare game. We first describe the network environment being defended, including the security measures and policies in place. We next present a scenario involving the red and blue players' objectives. We then describe the red players' attack moves and the resulting observables that are generated for the blue player by these actions. We step through the sequence of top-scoring COAs generated by Themistocles in the context of the moves undertaken by the red and blue players. We conclude with a summary of the results.

3.1 Cyber War Game Environment

We consider a network of 10 regular workstations that are used on a daily basis, as shown in the Themistocles screen capture of Fig. 5. Two additional workstations are used for system administration. There is also an internal web server used for organizational data sharing, an internal database server holding proprietary data shareable only with employees and an internal email server. At the network interface, there is a firewall with virtual private network (VPN) support. Outside the firewall, in the network demilitarized zone (DMZ), there is a network intrusion detection system (NIDS) machine with backup and archiving capabilities and a web server for sharing data with remote employees. An unknown number of external machines have valid access to the network.

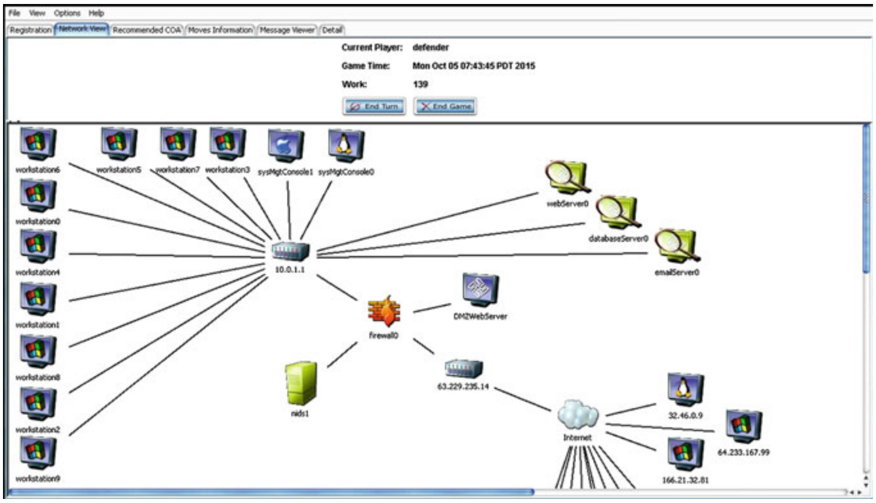

Current Player: defender
Game Time: Mon Oct 05 07:43:45 PDT 2015
Work: 139

Fig. 5 Themistocles screen shot showing the cyber warfare game environment

Certain security measures and policies have been implemented on the network. To wit, the fixed mission of the network allows the firewall to be configured to restrict incoming and outgoing traffic only to those services needed. Source information comes only via VPNs. The operating system (O/S) configurations have been stripped of unneeded functionality and services. All application software on the network machines is under configuration management. Finally the external IDS detects any attempts to violate the firewall rules.

3.2 Attack Scenario

The objectives of the red player in the game are to steal information from any computer on the network and to maintain access to the network for future attacks, while minimizing the chance of being discovered. We assume that the red player knows the IP address of the external web server in Fig. 5.

The objectives of the blue player are to maintain the availability of all servers while minimizing the amount of effort and resources required to do so, and to gather intelligence on any attacks in progress.

The attack proceeds as follows. Using a bot to help avoid attribution, the red player scans the DMZ web server for vulnerable services running on the machine. An exploit is used to gain root privileges on this server, ensuing with a backdoor installation. Using this backdoor, the red player maps the internal network. From this point, the red player attempts the exfiltration of data and the installation of root kits on any and all network devices that are exploitable. When the red player observes the failure of an attack against any particular host, a new host is selected for attack.

As a result of the red players' attack, the blue player observes the following on their control workstations:

- The IDS detects the port scan of the external web server. Due to background network activity, this observation does not lead to any response.
- A tripwire detects that new software has been installed on the external web server (i.e., the backdoor software).
- The IDS detects heavy download traffic to the external web server as the red player performs data exfiltration.
- A tripwire detects the upload of the rootkit software.

With these preliminaries, the blue player employs Themistocles to aid in her selection of moves to counter the attack in progress.

3.3 Themistocles Recommended Courses of Action

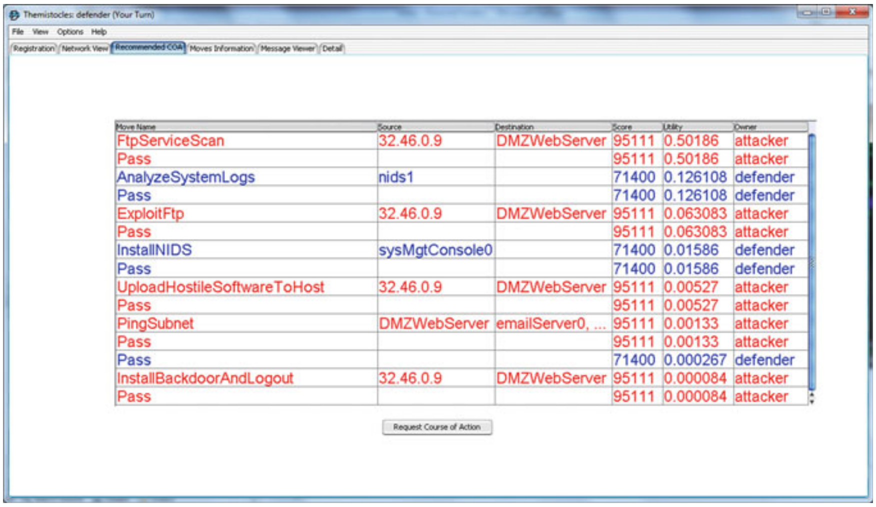
Table 6 shows the sequence of moves generated by Themistocles as the game proceeds from attack initiation to its conclusion. To illustrate the game, we will examine the first couple of COAs recommended by Themistocles for the blue player at the conclusion of the preceding sequence of moves by the red player.

Table 6 Sequence of moves by red and blue players

Move	Owner	Machine	Function
SetupExternalProxy	Attacker	externalHostX	
FTPScan	Attacker	externalHostX	DMZWebserver
AnalyzeSystemLogs	Defender	nids1	
Exploit FTP	Attacker	externalHostX	DMZWebserver
UploadHostileSoftware	Attacker	externalHostX	DMZWebserver
InstallBackdoor	Attacker	externalHostX	DMZWebserver
InstallNIDS	Attacker	sysMgtConsole	
LoginViaBackdoor	Attacker	externalHostX	DMZWebserver
ScanSubnetForVulnerabilities	Defender	sysMgtConsole	10.0.1.*
PingSubnetInternal	Attacker	DMZWebserver	10.0.1.*
PortScanSubnetInternal	Attacker	DMZWebserver	10.0.1.*
ExploitFtp	Attacker	DMZWebserver	Web server
UploadHostileSoftware	Attacker	DMZWebserver	Web server
HardenSystem	Defender	Web server	
InstallRootkit fails	Attacker	DMZWebserver	Web server
ExploitFtp	Attacker	DMZWebserver	DB server
ExfiltrateData	Attacker	DMZWebserver	DB server
DeployHoneypot	Attacker	DB server	
ModifyData	Attacker	DMZWebserver	DB server
ExfiltrateData fails	Attacker	DMZWebserver	DB server
ExploitFtp	Attacker	DMZWebserver	e-mail server
UploadHostileSoftware	Attacker	DMZWebserver	e-mail server
ApplyPatches	Defender	e-mail server	
InstallRootkit fails	Attacker	DMZWebserver	e-mail server
InstallRootkit	Attacker	externalHostX	DMZWebserver
Quarantine	Defender	DMZWebserver	

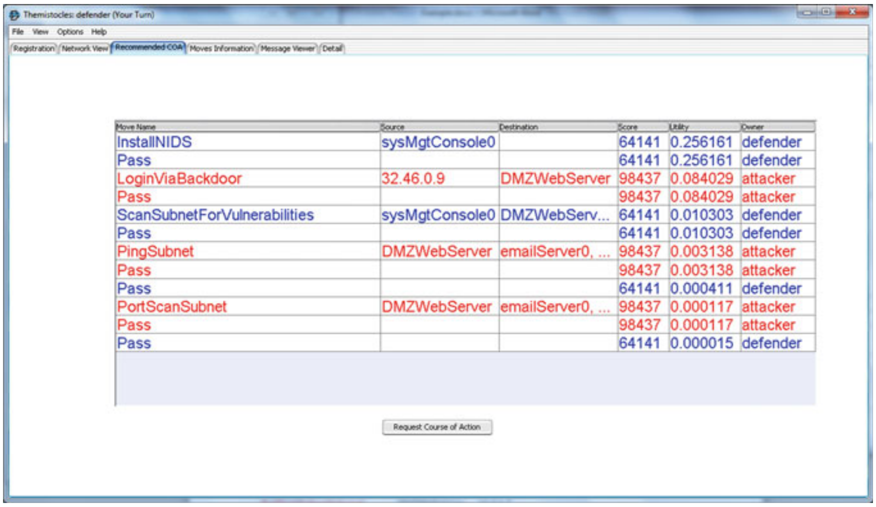
Given the red players' objectives, he begins with an FTP scan on the DMZ web server followed by a Pass. At the blue players' first turn, she observes the FTP scan and Themistocles analyzes the feasible subsequent moves and selects the highest-utility COA as shown in Fig. 6. Note that this COA includes moves by both the blue and red players, from the blue players' perspective of their own and the red players' scoring of these moves. The normalized utility of each successive move is shown in the next to last column. The depth of a COA tree is limited by setting the cutoff utility at 5×10^{-5}. Thus the COA in Fig. 6 represents the deepest COA having the highest utility above this cutoff threshold.

Referring back to the actual game moves in Table 6, the blue player elects to analyze the system logs as recommended for the next step by Themistocles. The red player then counters with the FTP exploit, uploading and installing the backdoor on the DMZ web server. When the blue players' next turn comes, Themistocles



Move Name	Source	Destination	Score	Utility	Owner
FtpServiceScan	32.46.0.9	DMZWebServer	95111	0.50186	attacker
Pass			95111	0.50186	attacker
AnalyzeSystemLogs	nids1		71400	0.126108	defender
Pass			71400	0.126108	defender
ExploitFtp	32.46.0.9	DMZWebServer	95111	0.063083	attacker
Pass			95111	0.063083	attacker
InstallNIDS	sysMgtConsole0		71400	0.01586	defender
Pass			71400	0.01586	defender
UploadHostileSoftwareToHost	32.46.0.9	DMZWebServer	95111	0.00527	attacker
Pass			95111	0.00527	attacker
PingSubnet	DMZWebServer	emailServer0, ...	95111	0.00133	attacker
Pass			95111	0.00133	attacker
Pass			71400	0.000267	defender
InstallBackdoorAndLogout	32.46.0.9	DMZWebServer	95111	0.000084	attacker
Pass			95111	0.000084	attacker

Fig. 6 Highest-utility COA for both players upon observation of the red FTP scan



Move Name	Source	Destination	Score	Utility	Owner
InstallNIDS	sysMgtConsole0		64141	0.256161	defender
Pass			64141	0.256161	defender
LoginViaBackdoor	32.46.0.9	DMZWebServer	98437	0.084029	attacker
Pass			98437	0.084029	attacker
ScanSubnetForVulnerabilities	sysMgtConsole0	DMZWebServ...	64141	0.010303	defender
Pass			64141	0.010303	defender
PingSubnet	DMZWebServer	emailServer0, ...	98437	0.003138	attacker
Pass			98437	0.003138	attacker
Pass			64141	0.000411	defender
PortScanSubnet	DMZWebServer	emailServer0, ...	98437	0.000117	attacker
Pass			98437	0.000117	attacker
Pass			64141	0.000015	defender

Fig. 7 Highest-utility COA for both players upon observation of the backdoor installation

generates the highest-utility COA shown in Fig. 7, whereupon the blue player follows the recommended next step and installs a NIDS on an internal router.

The red players' next step is to login via the backdoor software he has installed, as shown in Table 6. The game proceeds as shown by the moves in this table to the final step where the blue player quarantines the DMZ web server.

3.4 Analysis of the Example Game

The blue player began seeing suspicious behavior from alerts by its external NIDS early in the game. Themistocles recommended an installation of an additional NIDS on an internal router in order to catch malicious network activity if the red player had gotten inside the defenses. A vulnerability scan was recommended to determine if any unauthorized services had been installed. The backdoor installed on the DMZ web server was detected, and this forced the blue player to make an important decision—immediately remove the backdoor and make the DMZ web server unavailable to its regular users, or keep the server running and use this as an opportunity to identify the attacker. In keeping with the blue players' mission, Themistocles recommended the latter by deploying a "honeypot," while making sure to turn off all unnecessary services on the internal web server being attacked. The red player notices the web server is no longer available to attack, so heads for the database server. This leads the blue player to gather information on the red player, but the latter figures out that it is a honeypot when a data corruption attempt doesn't succeed. This is because honeypots do not have real data on them, so modifications aren't written to disk. The red player moves on to the e-mail server and at this point the blue player decides this is getting too aggressive and decides to shut out the red player by applying patches and quarantining the server to remove all malicious software.

In the end, the red player was able to gain access to the blue players' internal systems and map the entire internal network, but was not successful in stealing any data. One rootkit was successfully installed on the external DMZ web server, but that was a risk the blue player desired to take in order to gather more information on the red players' identity.

4 Conclusion

The Themistocles engine represents a well-tested application of game-theoretic principles to the cyber warfare domain. In several government-sponsored formal cyber war games, Themistocles has been shown to generate COAs for both offensive and defensive cyber warfare scenarios that are consistent with the move choices of independent experts monitoring the game.

Future work in this area will include the fuzzification of move scores to both type-1 and interval type-2 membership functions and the use of hierarchical linguistic weighted power means for the aggregation of COA scores (16–18). This will enable us to take account of the inherent imprecision associated with the costs/benefits of individual moves, and to employ a perspective ranging from the most pessimistic to the most optimistic on the aggregations of these scores.

References

Katz, A., Butler, B.: "Game Commander"-Applying an architecture of game theory and tree look ahead to the command and control process. In: Proceedings of the Fifth Annual Conference on AI, Simulation, and Planning (AIS94). Florida (1994)

Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM J. Res. Dev. **3**(3), 211–229 (1959)

Tesauro, G.: TD-Gammon, a Self-Teaching Backgammon Program, reaches master-level play. Neural Comput. **6**(2), 215–219 (1994)

Hsu, F., et al.: Deep thought. In: Marsland, T.A., Schaeffer, J. (eds.) Computer, Chess, and Cognition, pp. 55–78. Springer, New York (1990)

Hamilton, S.N., Hamilton, W.L.: Adversary modeling and simulation in cyber warfare. International Information Security Conference (2008)

Dudebout, N., Shamma, J.S.: Empirical evidence equilibria in stochastic games. In: 51st IEEE Conference on Deicsion and Control, December 2012 (2012)

Gopalakrishnan, R., Marden, J.R., Wierman, A.: An architectural view of game theoretic control. ACM SIGMETRICS Perform. Eval. Rev. **38**(3), 31–36 (2011)

Marden, J.R., Arslan, G., Shamma, J.S.: Connections between cooperative control and potential games. IEEE Trans. Syst. Man Cybern. Part B Cybern. **39**, 1393–1407 (2009)

Chen, L., Low, S.H., Doyle, J.C.: Random access game and medium access control design. IEEE/ACM Trans. Networking **18**(4), 1303–1316 (2010)

Chandra, F., Gayme, D.F., Chen, L., Doyle, J.C.: Robustness, optimization, and architectures. Eur. J. Control **5–6**, 472–482 (2011)

Chen, L., Li, N., Jiang, L., Low, S.H.: Optimal demand response: problem formulation and deterministic case. In: Chakrabortty, A., Ilic, M. (eds.) Control and Optimization Theory for Electric Smart Grids. Springer, New York (2012)

Carmel, D., Markovitch, S.: Learning and using opponent models in adversary search. Technical report CIS9606 (1996)

Meyers, K., Saydjari, O.S., et al.: ARDA cyber strategy and tactics workshop final report (2002)

Hamilton, S.N., Miller, W.L., Ott, A., Saydjari, O.S.: The role of game theory in information warfare. The information survivability workshop (2001)

Hamilton, S.N., Miller, W.L., Ott, A., Saydjari, O.S.: Challenges in applying game theory to the domain of information warfare. The information survivability workshop (2001)

Rickard, J.T., Aisbett, J.: New classes of threshold aggregation functions based upon the Tsallis q-exponential with applications to perceptual computing. Accepted for publication in IEEE Trans. Fuzzy Syst. (2013)

Rickard, J.T., Aisbett, J., Yager, R.R., Gibbon, G.: Linguistic weighted power means: comparison with the linguistic weighted average. In: Proceedings of FUZZ-IEEE 2011, 2011 World Congress on Computational Intelligence, June 2011, pp. 2185–2192. Taipei, Taiwan (2011)

Rickard, J.T., Aisbett, J., Yager, R.R., Gibbon, G.: Fuzzy weighted power means in evaluation decisions. In: Proceedings of World Symposium on Soft Computing, Paper #100, May 2011. San Francisco, CA (2011)