Game Strategies in Network Security

Kong-wei Lye[1] Jeannette Wing[2]

May 2002

CMU-CS-02-136

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

An extended abstract based on this report was submitted to the Foundations of Computer Security 2002, Copenhagen, Denmark.

Abstract

This paper presents a game-theoretic method for analyzing the security of computer networks. We view the interactions between an attacker and the administrator as a two-player stochastic game and construct a model for the game. Using a non-linear program, we compute the Nash equilibrium or best-response strategies for the players (attacker and administrator). We then explain why the strategies are realistic and how administrators can use these results to enhance the security of their network.

[1]Department of Electrical and Computer Engineering.
[2]Computer Science Department.

This research is sponsored in part by the Defense Advanced Research Projects Agency and the Army Research Office (ARO) under contract no. DAAD19-01-1-0485. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the DOD, ARO, or the U.S. Government.

Keywords: stochastic games, non-linear programming, network security

1 Introduction

Government agencies, schools, retailers, banks and a growing number of goods and service providers today all use the Internet as their integral way of conducting daily business. Individuals, good or bad, can also easily connect to the Internet. Due to the ubiquity of the Internet, computer security has now become more important than ever to organizations such as governments, banks, and businesses. Security specialists have long been interested in knowing what an intruder can do to a computer network, and what can be done to prevent or counteract attacks. In this paper, we describe how game theory can be used to find strategies for both an attacker and the administrator. We illustrate our approach with an example (Figure 1) of a local network connected to the Internet and consider the interactions between them as a general-sum stochastic game.

In Section 2, we introduce the formal model for stochastic games and relate the elements of this model to those in our network example. In Section 3, we explain the concept of a Nash equilibrium for stochastic games and explain what it means to the attacker and administrator. Then, in Section 4, we describe three possible attack scenarios for our network example. In these scenarios, an attacker on the Internet attempts to deface the homepage on the public web server on the network, launch an internal denial-of-service attack, and capture some important data from a workstation on the network. We compute the Nash equilibrium strategies (best responses) for the attacker and administrator using a non-linear program; we explain this solution for our example in Section 5. We discuss the implications of our approach in Section 6 and compare our work with previous work in the literature in Section 7. Finally, we summarize our results and point to directions for future work in Section 8.

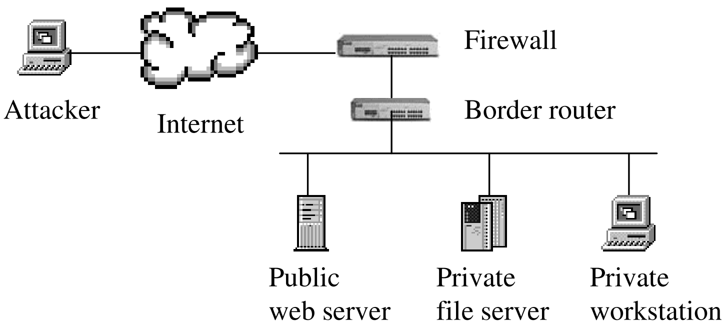

Figure 1: A Network Example

2 Networks as Stochastic Games

In this section, we first introduce the formal model of a *stochastic game*. We then use this model for our network attack example and explain how the state set, actions sets, cost/reward functions and transition probabilities can be defined or derived.

Formally, a two-player stochastic game is a tuple $(S, A^1, A^2, Q, R^1, R^2, \beta)$ where $S = \{\xi_1, \cdots, \xi_N\}$ is the state set and $A^k = \{\alpha_1^k, \cdots, \alpha_{M^k}^k\}$, $k = 1, 2$, $M^k = |A^k|$, is the action set of player k. The action set for player k at state s is a subset of A^k, i.e., $A_s^k \subseteq A^k$ and $\bigcup_{i=1}^N A_{\xi_i}^k = A^k$. $Q : S \times A^1 \times A^2 \times S \rightarrow$

$[0, 1]$ is the state transition function. $R^k : S \times A^1 \times A^2 \to \Re$, $k = 1, 2$ is the reward[1] function of player k. $0 < \beta \leq 1$ is a *discount factor* for discounting future rewards, i.e., at the current state, a state transition has a reward worth its full value, but the reward for the transition from the next state is worth β times its value at the current state.

The game is played as follows: at a discrete time instant t, the game is in state $s_t \in S$. Player 1 chooses an action a_t^1 from A^1 and player 2 chooses an action a_t^2 from A^2. Player 1 then receives a reward $r_t^1 = R^1(s_t, a_t^1, a_t^2)$ and player 2 receives a reward $r_t^2 = R^2(s_t, a_t^1, a_t^2)$. The game then moves to a new state s_{t+1} with conditional probability $\text{Prob}(s_{t+1}|s_t, a_t^1, a_t^2)$ equal to $Q(s_t, a_t^1, a_t^2, s_{t+1})$. In our example, we provide two views of the game: the attacker's view (Figure 3) and the administrator's view (Figure 4). These figures will be described in detail later in Section 4.

2.1 Network state

In general, the state of the network can contain various kinds of information or features such as type of hardware, software, connectivity, user privileges, etc. Using more features in the state allows us to represent the network better, but often makes the analysis more complex and difficult. We can view the network example as a graph (Figure 2). A node in the graph is a physical entity such as a workstation or router. We model the external world as a single computer (node E) and represent the web server, file server and workstation by nodes W, F and N, respectively. An edge in the graph represents a direct communication path (physical or virtual). For example, the external computer (node E) has direct access to only the public web server (node W).

Instantiating our game model, we let a superstate $< n_W, n_F, n_N, t > \in S$ be the state of the network. n_W, n_F and n_N are the *node states* for the web server, file server and workstation respectively, and t is a *traffic state* for the whole network. Each node X (where $X \in \{E, W, F, N\}$) has a node state $n_X =< P, a, d >$ to represent information about hardware and software configurations. $P \subseteq \{f, h, n, p, s, v\}$ is a list of software applications running on the node. We let f, h, n, and a denote *ftpd, httpd, nfsd* and some user process respectively; for malicious codes, s and v represent sniffer programs and viruses respectively. $a \in \{u, c\}$ is a variable used to represent the state of the user accounts. u represents normal user accounts and c means some user account has been compromised. We use the variable $d \in \{c, i\}$ to represent the state of the data on the node. c and i mean the data has and has not been corrupted or stolen respectively. For example, if $n_W =< (f, h, s), c, i >$, it means the web server is running an *ftpd* and an *httpd*; a sniffer program has been implanted; and a user account has been compromised but no data has been corrupted or stolen yet. The traffic information for the whole network is captured in a traffic state $t =< \{l_{XY}\} >$ where X and Y are nodes and $l_{XY} \in \{0, \frac{1}{3}, \frac{2}{3}, 1\}$ indicates the load carried on this link. A value of 1 indicates maximum capacity. For example, in a 10Base-T connection, the values 0, $\frac{1}{3}$, $\frac{2}{3}$ and 1 represent 0Mbps, 3.3Mbps, 6.7Mbps and 10Mbps respectively. In our example, the traffic state is $t =< l_{EW}, l_{WF}, l_{FN}, l_{NW} >$. We let $t =< \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3} >$ for normal traffic conditions.

The potential state space for our network example is very large but we shall discuss how to handle this problem in Section 6. The full state space in our example has a size of $|n_W| \times |n_F| \times |n_N| \times |t| = (63 \times 2 \times 2)^3 \times 4^4 \approx 4$ billion states but there are only 18 states (fifteen in Figure 3 and three additional ones in Figure 4) relevant to our illustration here. In these figures, each state is represented using a box with a symbolic state name and the values of the state variables. For convenience, we shall mostly refer to the states using their symbolic state names.

[1] We use the term "reward" in general here; in later sections, positive values are rewards and negative values are costs.

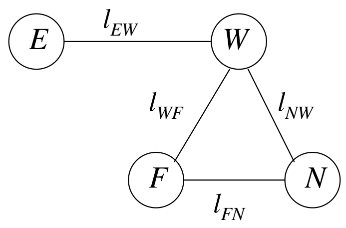

Figure 2: Network State

2.2 Actions

An action pair (one from the attacker and one from the administrator) causes the system to move from one state to another in a probabilistic manner. A single action for the attacker can be any part of his attack strategy, such as flooding a server with SYN packets or downloading the password file. When a player does nothing, we denote this inaction as ϕ. The action set for the attacker $A^{Attacker}$ consists of all the actions he can take in all the states, $A^{Attacker}$ = { *Attack_httpd*, *Attack_ftpd*, *Continue_hacking*, *Deface_website_leave*, *Install_sniffer*, *Run_DOS_virus*, *Crack_file_server_root_password*, *Crack_workstation_root_password*, *Capture_data*, *Shutdown_network*, ϕ}. His actions in each state is a subset of $A^{Attacker}$. For example, in the state **Normal_operation** (see Figure 3, topmost state), the attacker has an action set $A^{Attacker}_{\textbf{Normal\_operation}}$ = { *Attack_httpd*, *Attack_ftpd*, ϕ}.

Actions for the administrator are mainly taking preventive or restorative measures. In our example, the administrator has an action set $A^{Administrator}$ = { *Remove_compromised_account_restart_httpd*, *Restore_website_remove_compromised_account*, *Remove_virus_compromised_account*, *Install_sniffer_detector*, *Remove_sniffer_detector*, *Remove_compromised_account_restart_ftpd*, *Remove_compromised_account_sniffer*, ϕ}. In state **Ftpd_attacked** (see Figure 4), the administrator has an action set $A^{Adminstrator}_{\textbf{Ftpd\_attacked}}$ = { *install_sniffer_detector*, ϕ}.

A node with a compromised account may or may not be observable by the administrator. When it is not observable, we model the situation as the administrator having an empty action set in the state. We assume that the administrator does not know whether there is an attacker or not. Also, the attacker may have several objectives and strategies that the administrator does not know. Furthermore, not all of the attacker's actions can be observed.

2.3 State transition probabilities

In this example, we assign state transition probabilities based on intuition. In real life, case studies, statistics, simulations, and knowledge engineering can provide the required probabilities. In Figures 3 and 4, state transitions are represented by arrows. Each arrow is labeled with an action, a transition probability and a cost/reward. In the formal game model, a state transition probability is a function of both players' actions. Such probabilities are used in the non-linear program (Section 3) for computing a solution to the game. However, in order to separate the game into two views, we show the transitions as simply due to a single player's actions. For example, in Figure 3 (second dashed arrow from top), we show the derived probability Prob(**Ftpd_hacked** | **Ftpd_attacked**, *Continue_attacking*) = 0.5 as due to only the attacker's action *Continue_attacking*. When the network is in state **Normal_operation** and neither the attacker nor administrator takes any action, it will tend to stay in the same state. We model this situation as having a near-identity stochastic matrix, i.e., we let Prob(**Normal_operation** | **Normal_operation**, ϕ, ϕ)=1-ϵ for some small $\epsilon < 0.5$ and where ϕ denotes inaction. Then Prob(s | **Normal_operation**, ϕ, ϕ)= $\frac{\epsilon}{N-1}$ for all $s \neq$ **Normal_operation** where N is the number of states. There are also state transitions that are infeasible. For example, it may not be possible for the network to move from a normal operation state to a completely shutdown state without going through some intermediate states. Infeasible state transitions are assigned transition probabilities of 0.

2.4 Costs and Rewards

There are costs (negative values) and rewards (positive values) associated with the actions of the attacker and administrator. The attacker's actions have mostly rewards and such rewards are in terms of the amount of damage he does to the network. Some costs are, however, difficult to quantify. For example, the loss of marketing strategy information to a competitor can cause large monetary losses. A defaced corporate website may cause the company to lose its reputation and its customers to lose confidence.

In our model, we restrict ourselves to the amount of recovery effort (time) required by the administrator. The reward for an attacker's action is mostly defined in terms of the amount of effort the administrator has to make to bring the network from one state to another. For example, when a particular service crashes, it may take the administrator 10 or 15 minutes of time to determine the cause and restart the service [2]. In Figure 4, it costs the administrator 10 minutes to remove a compromised user account and to restart the *httpd* (from state **Httpd_hacked** to state **Normal_operation**). For the attacker, this amount of time would be his reward. To reflect the severity of the loss of the important financial data in our network example, we assign a very high reward for the attacker's action that leads to the state where the attacker gains this data. For example, from state **Workstation_hacked** to state **Workstation_data_stolen_1** in Figure 3, the reward is 999. There are also some transitions in which the cost to the administrator is not the same magnitude as the reward to the attacker. It is such transitions that make the game a general-sum game instead of a zero-sum game.

3 Nash Equilibrium

We now return to the formal model for stochastic games. Let $\Omega^n = \{p \in \Re^n \mid \sum_{i=1}^n p_i = 1, p_i \geq 0\}$ be the set of probability vectors of length n. $\pi^k : S \rightarrow \Omega^{M^k}$ is a stationary strategy for player k. $\pi^k(s)$ is the vector $[\pi^k(s,\alpha_1) \quad \cdots \quad \pi^k(s,\alpha_{M^k})]^{\mathrm{T}}$ where $\pi^k(s,\alpha)$ is the probability that player k should use to take action α in state s. A stationary strategy π^k is a strategy that is independent of time and history. A mixed or randomized stationary strategy is one where $\pi^k(s,\alpha) \geq 0$ $\forall s \in S$ and $\forall \alpha \in A^k$ and a pure strategy is one where $\pi^k(s,\alpha_i) = 1$ for some $\alpha_i \in A^k$.

The objective of each player is to maximize some expected return. Let s_t be the state at time t and r_t^k be the reward received by player k at time t. We define an expected return to be the column vector $v^k_{\pi^1,\pi^2} = [v^k_{\pi^1,\pi^2}(\xi_1) \quad \cdots \quad v^k_{\pi^1,\pi^2}(\xi_N)]^{\mathrm{T}}$ where

$$\begin{aligned} v^k_{\pi^1,\pi^2}(s) &= E_{\pi^1,\pi^2}\{r_t^k + \beta r_{t+1}^k + (\beta)^2 r_{t+2}^k + \cdots (\beta)^N r_{t+N}^k \mid s_t = s\} \\ &= E_{\pi^1,\pi^2}\{\sum_{n=0}^{N} (\beta)^n r_{t+n}^k \mid s_t = s\} \end{aligned}$$

[2]These numbers were given by the ECE Department's network manager.

The expectation operator $E_{\pi^1,\pi^2}\{\cdot\}$ is used to mean that player k plays π^k, i.e., player k chooses an action using the probability distribution $\pi^k(s_{t+n})$ at s_{t+n} and receives an immediate reward $r^k_{t+n} = \pi^1(s_{t+n})^{\mathrm{T}} R^k(s_{t+n})\pi^2(s_{t+n})$ for $n \geq 0$. $R^k(s) = [R^k(s, a^1, a^2)]_{a^1\in A^1, a^2\in A^2}$ [3], $k = 1, 2$ is player k's reward matrix in state s.

For an infinite-horizon game, we let $T = \infty$ and use a discount factor $\beta < 1$ to discount future rewards. $v^k(s)$ is then the expected total discounted rewards that player k will receive when starting at state s. For a finite-horizon game, $0 < T < \infty$ and $\beta = 1$. v^k is also called the *value vector* of player k.

A Nash equilibrium in stationary strategies (π^1_*, π^2_*) is one which satisfies:

$$v^1(\pi^1_*, \pi^2_*) \geq v^1(\pi^1, \pi^2_*) \qquad \forall \pi^1 \in \Omega^{M^1}$$

$$v^2(\pi^1_*, \pi^2_*) \geq v^2(\pi^1_*, \pi^2) \qquad \forall \pi^2 \in \Omega^{M^2}$$

component-wise. Here, $v^k(\pi^1, \pi^2)$ is the value vector of the game for player k when both players play their stationary strategies π^1 and π^2 respectively and $\geq$ is used to mean the left-hand-side vector is component-wise, greater than or equal to the right-hand-side vector. At this equilibrium, there is no mutual incentive for either one of the players to deviate from their equilibrium strategies π^1_* and π^2_*. A deviation will mean that one or both of them will have lower expected returns, i.e., $v^1(\pi^1, \pi^2)$ and/or $v^2(\pi^1, \pi^2)$. A pair of Nash equilibrium strategies is also known as best responses, i.e., if player 1 plays π^1_*, player 2's best response is π^2_* and vice versa.

In our network example, π^1 and π^2 corresponds to the attacker's and administrator's strategies respectively. $v^1(\pi^1, \pi^2)$ corresponds to the expected return for the attacker and $v^2(\pi^1, \pi^2)$ corresponds to the expected return for the administrator when they use the strategies π^1 and π^2. In a Nash equilibrium, when the attacker and administrator use their best-response strategies π^1_* and π^2_* respectively, neither will gain a higher expected return from using a different strategy if the other continues using his Nash strategy.

For infinite-horizon stochastic games, a non-linear program found in [FV96] can be used to find the stationary equilibrium strategies for both players and for finite-horizon games, a dynamic programming procedure, e.g., found in [FT91] can be used. For a thorough treatment on stochastic games, the reader is referred to [FV96].

A Nash equilibrium for a general-sum stochastic game can be found using the following non-linear program ([FV96]):

$$\min_{u^1,u^2,\sigma^1,\sigma^2} \left\{ \mathbf{1}^{\mathrm{T}} \left[u^k - R^k(\sigma^1, \sigma^2) - \beta P(\sigma^1, \sigma^2) u^k \right] \right\}, \qquad k = 1, 2$$

subject to:

$$R^1(\xi_i)\sigma^2(\xi_i) + \beta T(\xi_i, u^1)\sigma^2(\xi_i) \leq u^1(\xi_i)\mathbf{1}, \qquad i = 1, \cdots, N$$

$$\sigma^1(\xi_i)^{\mathrm{T}} R^2(\xi_i) + \beta\sigma^1(\xi_i) T(\xi_i, u^2) \leq u^2(\xi_i)\mathbf{1}^{\mathrm{T}}, \qquad i = 1, \cdots, N$$

where $u^k \in \Re^N$ are variables for value vectors and $\sigma^k \in \Omega^{M^k}$ are variables for strategies. $\mathbf{1}$ is a unit vector of appropriate dimensions.

[3] We use $[m(i,j)]_{i\in I, j\in J}$ to refer to an $|I| \times |J|$ matrix with elements $m(i,j)$.

$R^k(\pi^1, \pi^2)$ is the vector $[\pi^1(\xi_1)^{\mathrm{T}} R^k(\xi_1)\pi^2(\xi_1) \quad \cdots \quad \pi^1(\xi_N)^{\mathrm{T}} R^k(\xi_N)\pi^2(\xi_N)]^{\mathrm{T}}$ and $P(\pi^1, \pi^2)$ is the state transition probability matrix $[\pi^1(s)^{\mathrm{T}} \; [p(s' \mid s, a^1, a^2)]_{a^1 \in A^1, a^2 \in A^2} \; \pi^2(s)]_{s,s' \in S}$ induced by the strategy pair (π^1, π^2). $T(s, v) = [\; [p(\xi_1 \mid s, a^1, a^2) \cdots p(\xi_N \mid s, a^1, a^2)]^{\mathrm{T}} \; v \;]_{a^1 \in A^1, a^2 \in A^2}$ where v is an arbitrary value vector.

The two sets of constraints ($2 \times N$ inequalities) represent the optimality conditions required for the players and the global minimum to this non-linear program $(u^1_*, u^2_*, \sigma^1_*, \sigma^2_*)$ corresponds to a Nash solution $(v^1_*, v^2_*, \pi^1_*, \pi^2_*)$ of the game. Every general-sum discounted stochastic game has at least one Nash equilibrium in stationary strategies (see [FV96]) (not necessarily unique) and finding these equilibria is non-trivial.

In our network example, finding multiple Nash equilibria means finding multiple pairs of Nash strategies. In each pair, a strategy for one player is a best-response to the strategy for the other player and vice versa. We shall refer to the non-linear program described in this section as NLP-1 and use it to find the Nash equilibrium for our network example later in Section 5.

4 Attack and Response Scenarios

In this section, we describe three different attack and response scenarios. We show in Figure 3 how the attacker sees the state of the network change as a result of his actions. Figure 4 shows the administrator's viewpoint. In both figures, a state is represented using a box containing the symbolic name and the values of the state variables for that state. Each transition is labeled with an action, the probability of the transition, and the gain or cost in minutes of restorative effort incurred on the administrator. The three scenarios are indicated using bold, dotted and dashed arrows in Figure 3. Due to space constraints, not all state transitions for every action are shown. From one state to the next, state variable changes are highlighted using bold font.

4.1 Scenario 1 (Bold)

A common target for use as a launching base in an attack is the public web server. The web server typically runs an *httpd* and an *ftpd* and a common technique for the attacker to gain a root shell is *buffer overflow*. Once the attacker gets a root shell, he can deface the website and leave. This scenario is shown by the state transitions indicated by bold arrows in Figure 3.

From state **Normal_operation**, the attacker takes action *Attack_httpd*. With a probability of 1.0 and a reward of 10, he moves the system to state **Httpd_attacked**. This state indicates increased traffic between the external computer and the web server as a result of his attack action. Taking action *Continue_attacking*, he has a 0.5 probability of success of gaining a user or root access through bringing down the *httpd*, and the system moves to state **Httpd_hacked**. Once he has root access in the web server, he can deface the website, restart the *httpd* and leaves, moving the network to state **Website_defaced**.

4.2 Scenario 2 (Dotted)

The other thing that the attacker can do after he has hacked into the web server is to launch a *denial-of-service* (DOS) from inside the network. This is shown by the state transitions drawn using dotted arrows (starts from the middle of Figure 3), with each state having more internal traffic than the previous. From state **Webserver_sniffer**, the attacker takes action *Run_DOS_virus*. With probability 1 and a reward of 30, the network moves into state **Webserver_DOS_1**. In this state, the traffic load on all internal links has increased from $\frac{1}{3}$ to $\frac{2}{3}$. From this state, the network degrades to state **Webserver_DOS_2** with probability 0.8 even when the attacker does nothing. The traffic load is now at full capacity of 1 in all the links. We assume that there is a 0.2 probability that the administrator notices this and takes action to recover the system. In the very last state, the network grinds to a halt and nothing productive can take place.

4.3 Scenario 3 (Dashed)

Once the attacker has hacked into the web server, he can install a sniffer and a backdoor program. The sniffer will sniff out passwords from the users in the workstation when they access the file server or web server. Using the backdoor program, the attacker then comes back to collect his password list from the sniffer program, cracks the root password and logs on to the workstation and search the local hard disk. This scenario is shown by the state transitions indicated by dashed arrows in Figure 3. From state **Normal_operation**, the attacker takes action *Attack_ftpd*. With a probability of 1.0 and a reward of 10, he moves the system to state **Ftpd_attacked**. There is increased traffic between the external computer and the web server as well as between the web server and the file server in this state, both from $\frac{1}{3}$ to $\frac{2}{3}$. If he continues to attack the *ftpd*, he has a 0.5 probability of success of gaining a user or root access through bringing down the *ftpd*, and the system moves to state **Ftpd_hacked**. From here, he can install a sniffer program and with probability 0.5 and a reward of 10, move the system to state **Webserver_sniffer**. In this state, he has also restarted the *ftpd* to avoid causing suspicion from normal users and the administrator. The attacker then collects the password list and cracks the root password on the workstation. We assume he has a 0.9 chance of success and when he succeeds, he gains a reward of 50 and moves the network to state **Workstation_hacked**. To cause more damage to the network, he can even shut it down using the privileges of root user in this workstation.

4.4 Recovery

We now turn our attention to the administrator's view (see Figure 4). The administrator in our example does mainly restorative work and his actions can be restarting the *ftpd*, removing a virus, etc. He also takes preventive measures and such actions can be installing a sniffer detector, re-configuring a firewall, deactivating a user account, and so on. In the first attack scenario in which the attacker defaces the website, the administrator can only take the action *Restore_website_remove_compromised_account* to bring the network from state **Website_defaced** to **Normal_operation**. In the second attack scenario, the states **Webserver_DOS_1** and **Webserver_DOS_2** (indicated by double boxes) show the network suffering from the effects of the internal DOS attack. All the administrator can do is take the action *Remove_virus_compromised_account* to bring the network back to **Normal_operation**. In the third attack scenario, there is nothing he can do to restore the network back to its original operating state. The important data has been stolen and there is no way he can undo this. The network can only move from state **Workstation_data_stolen_1** to **Workstation_data_stolen_2** (indicated by dotted box on bottom right in Figure 4).

The state **Ftpd_attacked** (dashed box) is an interesting state because here, the attacker and administrator can engage in real-time game play. In this state, when the administrator notices an unusual increase in traffic between the external network and the web server and also between the web server and the file server, he may suspect an attack is going on and takes action *Install_sniffer_detector*. Taking this action, however, incurs a cost of 10. If the attacker is still attacking, the system moves into state **Ftpd_attacked_detector**. If he has already hacked into the web server, then the system moves to state **Webserver_sniffer_detector**. Detecting the sniffer program, the administrator can now remove the affected user account and the sniffer program to prevent the attacker from further attack actions.

5 Results

We coded NLP-1 in *MATLAB* (a mathematical computation software package by *The MathWorks, Inc.*) and use it a Nash equilibrium solution for our example described in Section 4. There could be several equilibria in the game but we shall discuss the only one we found. To run 3, the cost/reward and state transition functions defined in Section 2 are required. In our formal game model, the state of the game evolves only at discrete time instants. This requirement is not relevant in our example and is ignored. The game model also requires actions to be taken simultaneously by both players. There are some states in which a player has only one or two non-trivial actions and for consistency and easier computation using 3, we add an *inaction* ϕ to the action set for the state so that the action sets are all of the same cardinality. Overall, our game model has 18 states and 3 actions per state and the corresponding non-linear program takes approximately 45 minutes to run on a computer equipped with a 600Mhz Pentium-III and 128Mb of RAM.

The result of NLP-1 is a Nash equilibrium. It consists of a pair of strategies ($\pi_*^{Attacker}$ and $\pi_*^{Administrator}$) and a pair of value vectors ($v_*^{Attacker}$ and $v_*^{Administrator}$) the attacker and administrator. The strategy for a player consists of a probability distribution over the action set for each state. The Nash equilibrium strategies and value vectors are shown in Table 1 and Table 2 respectively.

	State	Attacker's Strategy	Administrator's Strategy
1	**Normal_operation**	[1.00 0.00 0.00]	[0.33 0.33 0.33]
2	**Httpd_attacked**	[1.00 0.00 0.00]	[0.33 0.33 0.33]
3	**Ftpd_attacked**	[0.65 0.00 0.35]	[1.00 0.00 0.00]
4	**Ftpd_attacked_detector**	[0.40 0.12 0.48]	[0.93 0.07 0.00]
5	**Httpd_hacked**	[0.33 0.10 0.57]	[0.67 0.19 0.14]
6	**Ftpd_hacked**	[0.12 0.00 0.88]	[0.96 0.00 0.04]
7	**Website_defaced**	[0.33 0.33 0.33]	[0.33 0.33 0.33]
8	**Webserver_sniffer**	[0.00 0.50 0.50]	[0.33 0.33 0.34]
9	**Webserver_sniffer_detector**	[0.34 0.33 0.33]	[1.00 0.00 0.00]
10	**Webserver_DOS_1**	[0.33 0.33 0.33]	[1.00 0.00 0.00]
11	**Webserver_DOS_2**	[0.34 0.33 0.33]	[1.00 0.00 0.00]
12	**Network_shut_down**	[0.33 0.33 0.33]	[0.33 0.33 0.33]
13	**Fileserver_hacked**	[1.00 0.00 0.00]	[0.35 0.34 0.31]
14	**Fileserver_data_stolen_1**	[1.00 0.00 0.00]	[1.00 0.00 0.00]
15	**Workstation_hacked**	[1.00 0.00 0.00]	[0.31 0.32 0.37]
16	**Workstation_data_stolen_1**	[1.00 0.00 0.00]	[1.00 0.00 0.00]
17	**Fileserver_data_stolen_2**	[0.33 0.33 0.33]	[0.33 0.33 0.33]
18	**Workstation_data_stolen_2**	[0.33 0.33 0.33]	[0.33 0.33 0.33]

Table 1: Nash equilibrium strategies for attacker and administrator

We explain the strategies for some of the more interesting states here. For example, in the

	State	Attacker's State Value	Administrator's State Value
1	**Normal_operation**	210.2	-206.8
2	**Httpd_attacked**	202.2	-191.1
3	**Ftpd_attacked**	176.9	-189.3
4	**Ftpd_attacked_detector**	165.8	-173.8
5	**Httpd_hacked**	197.4	-206.4
6	**Ftpd_hacked**	204.8	-203.5
7	**Website_defaced**	80.4	-80.0
8	**Webserver_sniffer**	716.3	-715.1
9	**Webserver_sniffer_detector**	148.2	-185.4
10	**Webserver_DOS_1**	106.7	-106.1
11	**Webserver_DOS_2**	96.5	-96.0
12	**Network_shut_down**	80.4	-80.0
13	**Fileserver_hacked**	1065.5	-1049.2
14	**Fileserver_data_stolen_1**	94.4	-74.0
15	**Workstation_hacked**	1065.5	-1049.2
16	**Workstation_data_stolen_1**	94.4	-74.0
17	**Fileserver_data_stolen_2**	80.4	-80.0
18	**Workstation_data_stolen_2**	80.4	-80.0

Table 2: Nash equilibrium state values for attacker and administrator

state **Httpd_hacked** (5^{th} row in Table 1), the attacker has action set { *Deface_website_leave, Install_sniffer,* ϕ }. His strategy for this state says that he should use *Deface_website_leave* with probability 0.33 and *Install_sniffer* with probability 0.10. Ignoring the last action ϕ, and after normalizing, these probabilities become 0.77 and 0.23 respectively for *Deface_website_leave* and *Install_sniffer*. Even though installing a sniffer may allow him to crack a root password and eventually capture the data he wants, there is also the possibility that the system administrator detects his presence and takes preventive measures. He is thus able to do more damage (probabilistically speaking) if he simple defaces the website and leaves. In this same state, the administrator can either take action *Remove_compromised_account_restart_httpd* or *Install_sniffer_detector*. His strategy says that he should take the former with probability 0.67 and the latter with probability 0.19. Ignoring the third action ϕ and after normalizing, these probabilities become 0.88 and 0.22 respectively. This tells him that he should immediately remove the compromised account and restart the *httpd* rather than continue to "play" with the attacker. It is not shown here in our model but installing the sniffer detector could be a step towards apprehending the attacker, which means greater reward for the administrator. In the state **Webserver_sniffer** (8^{th} row in Table 1), the attacker should take the actions *Crack_file_server_root_password* and *Crack_workstation_root_password* with equal probability (0.5) because either action will let him do the same amount of damage eventually. Finally, in the state **Webserver_DOS_1** (10^{th} row in Table 1), the system administrator should remove the DOS virus and compromised account, this being his only action in this state (the other two being ϕ).

In Table 2, we note that the value vector for the administrator is not exactly the negative of that for the attacker because in our example, not all state transitions have costs and rewards that are of the same magnitude. In a zero-sum game, the value vector for one player is the negative of the other's. In this table, the negative state values for the administrator correspond to his expected costs or expected amount of recovery time (in minutes) required to bring the network back to normal operation. Positive state values for the attacker correspond to his expected reward or the expected amount of damage he causes to the administrator (again, in minutes of recovery time). Both the attacker and administrator would want to maximize the state values for all the states. In state **Fileserver_hacked** (13^{th} row in Table 2), the attacker has gained access into the file server and has full control over the data in it. In state **Workstation_hacked** (15^{th} row in Table 2), the attacker has gained root access to the workstation. These two states have the same value of 1065.5, the highest among all states, because these are the two states that will lead him to the greatest damage to the network. When at these states, the attacker is just one state away from capturing the desired data from either the file server or the workstation. For the administrator, these two states have the most negative values (-1049.2), meaning most damage can be done to his network when it is in either of these states. In state **Webserver_sniffer** (8^{th} row in Table 2), the attacker has a state value of 716.3, which is relatively high compared to those for other states. This is the state in which he has gained access to the public web server and installed a sniffer, i.e., a state that will potentially lead him to stealing the data that he wants. At this state, the value is -715.1 (second most negative value) for the administrator. This is the second least desirable state for him.

6 Discussion

We could have modeled the interaction between the attacker and administrator as a purely competitive (zero-sum) stochastic game, in which case we would always find only a single unique Nash equilibrium. Modeling it as a general-sum stochastic game however, allows us to find potentially, multiple Nash equilibria. A Nash equilibrium gives the administrator an idea of the attacker's strategy and a plan for what to do in each state in the event of an attack. Finding more Nash equilibria thus allows him to know more about the attacker's best attack strategies. By using a stochastic game model, we are also able to capture the probabilistic nature of the state transitions, which is more realistic. Solutions for stochastic models are however, hard to compute.

A disadvantage of our model is that the full state space can be extremely large. We are however, interested in only a small subset of states that are in attack scenarios. One way of generating these states is the attack scenario generation method developed by Sheyner et al [SJW02]. The set of scenario states can then be augmented with state transition probabilities and costs/rewards as functions of both players' actions so that our game-theoretic analysis can be applied. Another difficulty in our analysis is in building the game model. In reality, it may be difficult to quantify the costs for some actions and transition probabilities may not be easily available.

We note that the administrator's view of the game in our example is simplistic and uninteresting. This is because he only needs to act when he suspects the network is under attack. It is reasonable to assume the attacker and administrator both know what the other can do. Such common knowledge affects their decisions on what to do in each state and thus justifies a game formulation of the problem.

One may argue why not put in place all security measures. In practice, trade-offs have to be made between security and usability and a network may have to remain in operation despite known vulnerabilities (e.g., see [Cru00]). Knowing that a network system is not perfectly secure, our game theoretic formulation of the security problem allows the administrator to discover the potential attack strategies of an attacker as well as best defense strategies against them.

7 Related Work

The use of game theory in modeling good and evil has also appeared in several other areas of research. For example, in military and information warfare, the enemy is modeled as an evil player and has actions and strategies to disrupt the defense networks. Browne describes how static games can be used to analyze attacks involving complicated and heterogeneous military networks [Bro00]. In his example, a defense team has to defend a network of three hosts against an attacking team's *worms*. A defending team member can choose either to run a worm detector or not. Depending on the combined attack and defense actions, each outcome has different costs. This problem is similar to ours if we were to view the actions of each team member as separate actions of a single player. The interactions between the two teams are dynamic, however, and can be better represented using a stochastic model like we did here. In his Master's thesis, Burke studies the use of repeated games with incomplete information to model good and evil players in information warfare [Bur99]. As in our work, the objective is to predict enemy strategies and find defenses against them using a game model. Using static game models, however, requires the problem to be abstracted to a very high level and only simple analyses are possible. Our use of a stochastic model in this paper allows us to capture the probabilistic nature of state transitions in real life.

In the study of network reliability, Bell considers a zero-sum game in which the router has to find a least-cost path and a network tester seeks to maximize this cost by failing a link [Bel01]. The problem is similar to ours in that two players are in some form of control over the network and that they have opposite objectives. Finding the least-cost path in their problem is analogous to finding a best defense strategy in ours. Hespanha and Bohacek discusses routing games in which an adversary tries to intersect data packets in a computer network [HB01]. The designer of the network has to find routing policies that avoid links that are under the attacker's surveillance. Finding their optimal routing policy is similar to finding the least-cost path in Bell's work [Bel01] and the best defense strategy in our problem in that at every state, each player has to make a decision on what action to take. Their game model is, again, a zero-sum game. In comparison, our work uses a more general (general-sum) game model that allows us to find more Nash equilibria.

McInerney et al. use a simple one-player game in their *FRIARS* cyber-defense decision system capable of reacting autonomously to automated system attacks [MSAH01]. Their problem is similar to ours in that good is fighting evil in cyberspace. Instead of finding complete strategies, their single-player game model is used to predict the opponent's next move one at a time. Their model is closer to being just a Markov decision problem because it is a single-player game. Ours, in contrast, exploits fully what a game (two-player) model can allow us to find, namely, equilibrium strategies for both players.

Finally, Syverson talks about "good" nodes fighting "evil" nodes in a network and suggested using stochastic games for reasoning and analysis [Syv97]. We have given precisely such an example in this paper. In summary, our work and example is different from previous work in that we employ a general-sum stochastic game model. This allows us to perform a richer analysis for more complicated problems and also allows us to find multiple Nash equilibria (sets of best responses) instead of a single equilibrium.

8 Conclusion

We have shown how the network security problem can be modeled as a general-sum stochastic game between the attacker and the administrator. Using the non-linear program NLP-1, we computed the Nash equilibria (strategies and value vectors) for both the players and then explained why these strategies make sense and are useful for the administrator. Discussions with one of our university's network managers revealed that these results are indeed useful. With proper modeling, the game-theoretic analysis we presented here can also be applied to other general heterogeneous networks.

In the future, we wish to develop a systematic method for decomposing large models into smaller manageable components such that strategies can be found individually for them using conventional Markov decision process (MDP) and game-theoretic solution methods such as dynamic programming, policy iteration and value iteration. For example, nearly-isolated clusters of states can be regarded as subgames and states in which only one player has meaningful actions can be regarded as an MDP. The overall best-response for each player is then composed from the strategies for the components. It is believed that the computation time can be significantly reduced by using such a decomposition method. We also intend to use the method by Sheyner et al [SJW02] for attack scenario generation to generate states so that we can experiment with network examples that are larger and more complicated. In our example, we manually enumerated the states for the attack scenario. The method in [SJW02] allows us to generate automatically the complete set of attack scenario states and thus allows us to perform a more complete analysis.

References

[Bel01] M.G.H. Bell. The measurement of reliability in stochastic transport networks. *Proceedings, 2001 IEEE Intelligent Transportation Systems*, pages 1183–1188, 2001.

[Bro00] R. Browne. C4I defensive infrastructure for survivability against multi-mode attacks. In *Proceedings, 21st Century Military Communications. Architectures and Technologies for Information Superiority*, volume 1, pages 417–424, 2000.

[Bur99] David Burke. Towards a game theory model of information warfare. Master's thesis, Graduate School of Engineering and Management, Airforce Institute of Technology, Air University, 1999.

[Cru00] Jeff Crume. *Inside Internet Security.* Addison Wesley, 2000.

[FT91] Drew Fudenberg and Jean Tirole. *Game Theory.* MIT Press, 1991.

[FV96] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes.* Springer-Verlag, New York, 1996.

[HB01] J.P Hespanha and S. Bohacek. Preliminary results in routing games. In *Proceedings, 2001 American Control Conference*, volume 3, pages 1904–1909, 2001.

[MSAH01] J. McInerney, S. Stubberud, S. Anwar, and S. Hamilton. Friars: a feedback control system for information assurance using a markov decision process. In *Proceedings, IEEE 35th Annual 2001 International Carnahan Conference on Security Technology*, pages 223–228, 2001.

[SJW02] O. Sheyner, S. Jha, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 2002.

[Syv97] Paul F. Syverson. A different look at secure distributed computation. In *Proceedings, 10th Computer Security Foundations Workshop*, pages 109–115, 1997.

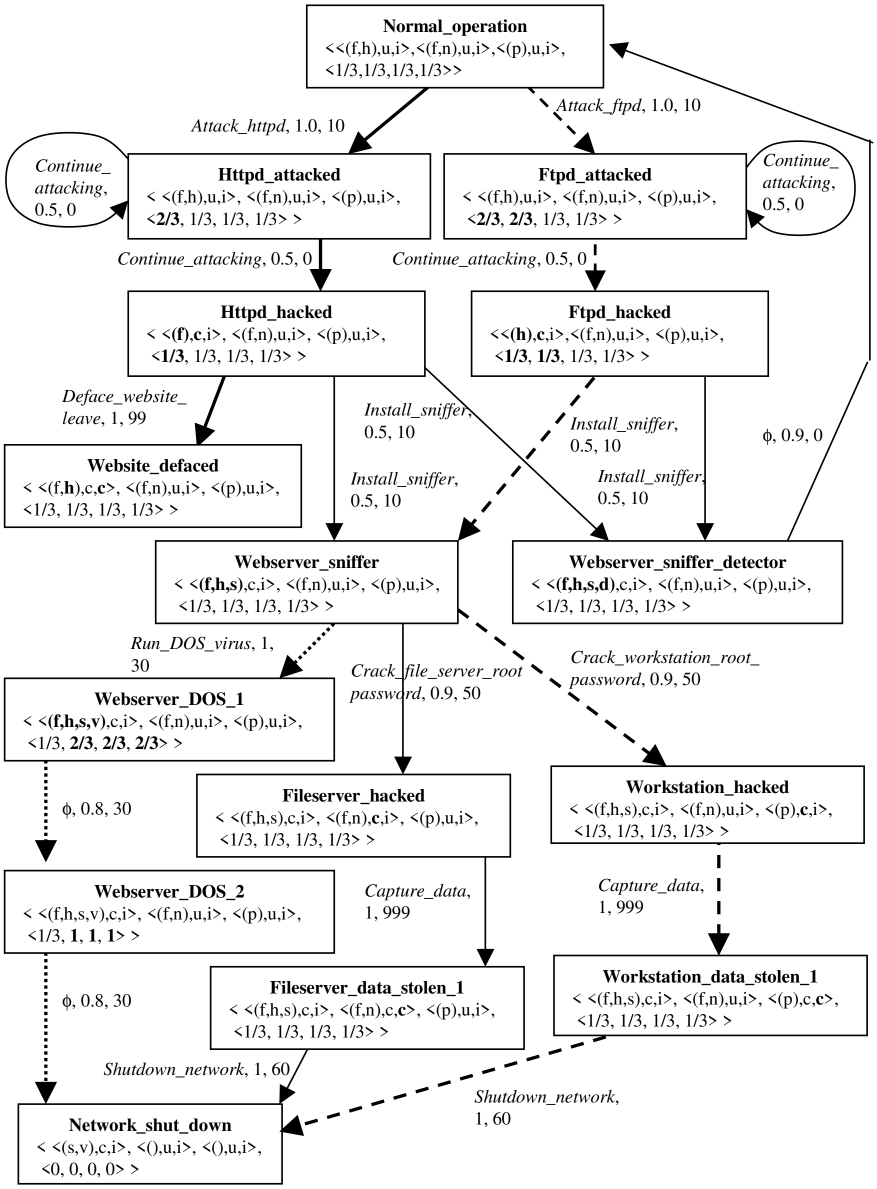

Figure 3: Attacker's view of the game

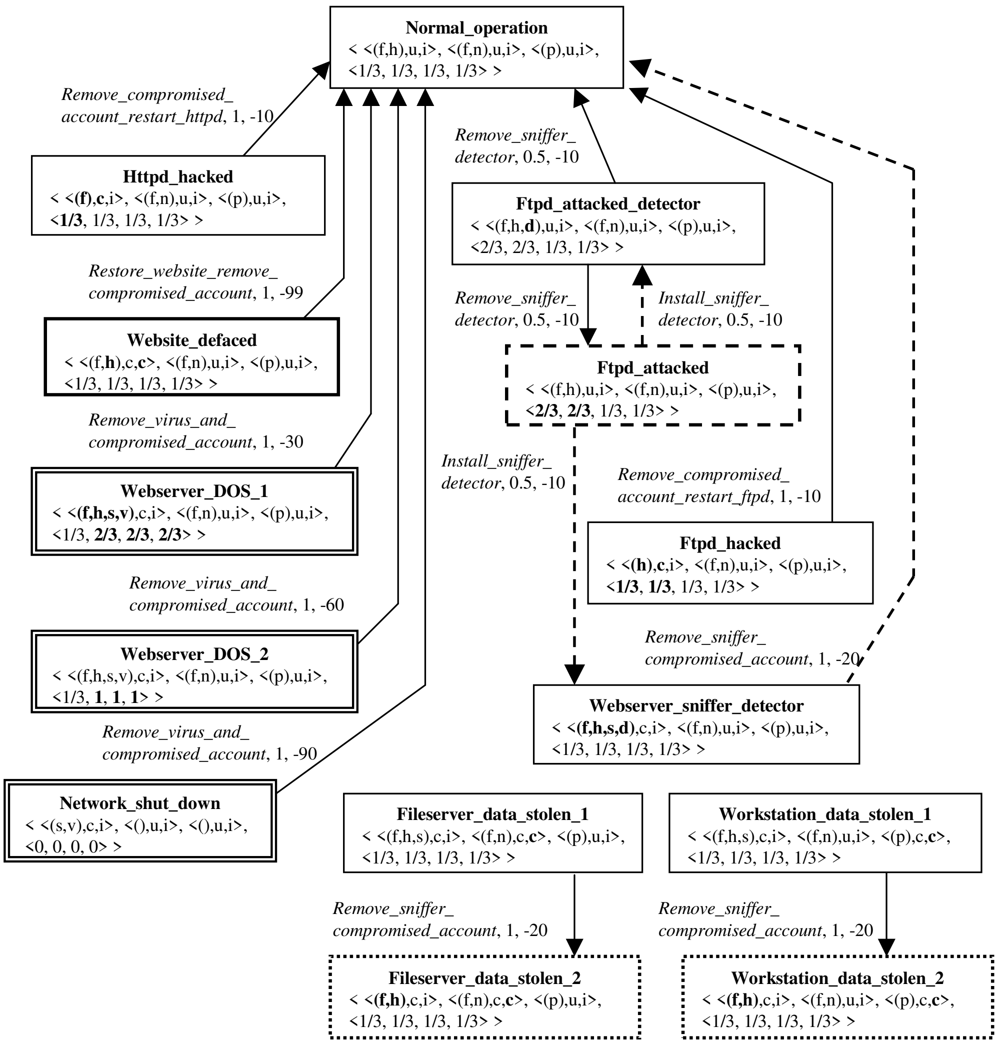

Figure 4: Administrator's view of the game