Journal of King Saud University – Computer and Information Sciences (2015) 27, 386–392

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

# Optimization of rootkit revealing system resources – A game theoretic approach

**K. Muthumanickam** *, **E. Ilavarasan**

*Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry 605 014, India*

Received 22 January 2014; revised 10 June 2014; accepted 23 October 2014
Available online 10 September 2015

**KEYWORDS**

Computer security;
Non-cooperative game theory;
Rootkit;
Resource optimization;
Windows OS

**Abstract** Malicious rootkit is a collection of programs designed with the intent of infecting and monitoring the victim computer without the user's permission. After the victim has been compromised, the remote attacker can easily cause further damage. In order to infect, compromise and monitor, rootkits adopt Native Application Programming Interface (API) hooking technique. To reveal the hidden rootkits, current rootkit detection techniques check different data structures which hold reference to Native APIs. To verify these data structures, a large amount of system resources are required. This is because of the number of APIs in these data structures being quite large. Game theoretic approach is a useful mathematical tool to simulate network attacks. In this paper, a mathematical model is framed to optimize resource consumption using game-theory. To the best of our knowledge, this is the first work to be proposed for optimizing resource consumption while revealing rootkit presence using game theory. Non-cooperative game model is taken to discuss the problem. Analysis and simulation results show that our game theoretic model can effectively reduce the resource consumption by selectively monitoring the number of APIs in windows platform.



## 1. Introduction

Today, approximately 90% of the operating systems in the Internet run windows operating system (W3Schools). This enables the remote attacker to easily damage many computer systems after getting an entry point in a victim computer. Malicious rootkits refer to a collection of software routines designed to hide their presence and other malicious activities and enable the attacker to take control of the victim computer (Emigh, 2006). Moreover, rootkits can also be used as backdoor to spy user or system's activities (Quynh and Take Fuji, 2007). The attacker can then capture sensitive information about either end-user or computer. As 85% of malicious software is being developed today with the intention of affecting windows operating system to harvest money (Wang and Dasgupta, 2007), we focus on the area of windows rootkit detection. In order to launch malicious activities, windows rootkits adopt a mechanism called 'hooking' which can modify the predefined execution path of a system call. However,

* Corresponding author.
E-mail addresses: kmuthoo@pec.edu (K. Muthumanickam), eilavarasan@pec.edu (E. Ilavarasan).

Peer review under responsibility of King Saud University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2014.10.004

rootkits need to access Native APIs to accomplish their tasks. There have been many approaches proposed in the past which rely on signature-based method. There were some effective anti-rootkit tools also available to dynamically analyze the behavior of Rootkit Malwares. However, they failed to detect native API hooks dynamically. Therefore, finding a mechanism that is capable of detecting malicious Native API hook rootkits to prevent both user-space and kernel-space is a challenging problem. To analyze the economic aspects of windows malicious rootkit activity and optimize the resource to be monitored, we investigate a game theoretical approach that models the relationship between the defender and the attacker. Our game theoretic approach will guide the defender to have optimal resources to reveal the presence of a rootkit. In order to attain the ultimate goal, an attacker might maximize his protection by utilizing maximum resources in the worst case whereas the defender's goal might be to optimize rootkit detection by resource consumption, i.e., by monitoring minimal number of APIs. When a Rootkit Revealing Module (RRM) is running on a host computer, it is necessary to continuously monitor more number of user-space objects and kernel-space objects, regardless of the possibility of an attack. To deal with this issue, we propose a game theoretical approach to reduce the monitoring APIs being monitored by RRM without loss of its detection accuracy. The monitored APIs are chosen according to the amount of resource consumed and the expected largest probability to attack. A game theoretic model to the Intrusion Detection System assists in the decision process of allocating limited resource for detecting significant threats in a large network in linear time. So, we have chosen a game theoretic approach to model the interaction between RRM and Rootkit Malware (RM) API hook attack to find the optimal number of APIs to be monitored. Here the two players are: the RRM and the RM. In this model, the RRM can choose to monitor the system or not, and for how long to monitor. On the other hand, the RM can choose to attack or not, or delay the attacking time to evade the RRM. As computer attacks are launched repeatedly, we select a repeated non-cooperative game model. The final outcome guides the RRM to use minimal number of APIs with respect to the attack scenarios.

The rest of the paper is aligned as follows: Section 2 discusses related work. Section 3 defines the problem statement. We explain a game theory model in Section 4. Furthermore, a case study is presented in Section 5. Section 6 presents the simulation results. Finally, in Section 7 we conclude the proposed work.

## 2. Related works

As malware writers have devised new methods to violate computer security policies, many researchers focused developing a new technique to combat them. An intrusion system based on Bayesian probability has been proposed (Altwaijry and Algarny, 2012) in which naïve Bayesian classifier is mainly used to identify four different classes of attacks. The system was trained using KDD data set to achieve better detection rate. The authors of Abdullah Al-Kadhi (2011) proposed an assessment report on spam in the Kingdom of Saudi Arabia. The study paper also discussed about anti-spam efforts in different countries and emerging anti-spam technologies. The paper provides a basis for researchers who look for knowledge in spam type of attack. Another approach (Alfantookh, 2006), particularly for detecting DoS attacks was proposed. Their system used the idea of neural network to classify known and unknown attacks from network traffic packets and achieved better results. In this context, only few works addressed the issue of resource optimization. Today, game theory is being effectively applied to address many real world issues. There have been plenty of ideas proposed in the field of computer security especially in the area of Intrusion Detection Systems (IDS). But limited methods existed to model malicious rootkit detection in windows platform using game theory. Game theory gained authenticity because of John Von Neumann and Morgenstern and they published a book in 2004 (Neumann et al., 2004). Thereafter, it has been applied in the fields of biology, economics, sociology, etc (Game theory). A game is played between two or more players with different strategies. A payoff/reward has been awarded to the player for each and every action within the game. The payoff may be either positive or negative value. The game solution guides each player to know their optimal strategy against the opponent.

Intrusion detection has long been an active research topic in the detection of potential attacks. There have been limited approaches addressing the use of game theory to improve the performance of the detection module. Liu et al. (2002) presented a game model to optimize intrusion detection strategies in a closed network. In Liu (2005), authors discussed a game theoretic approach to predict cyber attacks. In Kodialam and Lakshman (2003), authors developed a game theoretic framework to formulate the game interaction between the intruder and the service provider. The optimal strategy of the intruder is to minimize the probability of being detected and the service provider's objective is to maximize the detection probability. In Alpcan and Baskar (2004), the authors modeled a non-cooperative and non-zerosum game to discuss continuous-kernel version. The authors proved the existence of the Nash Equilibrium and discussed the dynamics of the game. Few approaches (Pacha and Park, 2006; Liu et al., 2006) have been proposed over ad-hoc networks. Liu et al. (2008) proposed a non-cooperative game model to enable the Host Intrusion Detection System to optimize the resources to be monitored. Also, a multi-stage buffer overflow attack was taken as a case study and it was concluded that their model utilizes minimal objects. In Chen and Leneutre (2009), authors addressed the intrusion detection problem in heterogeneous networks using game theoretic approach. They discussed the problem as a non-cooperative game between the attacker and the defender. To achieve optimal system value, they derived optimal strategy for the defender and minimal resource consumption. Otrok et al. (2008), presented game theory model to discuss the issues of detecting intrusions in wired network. A sampling strategy has been derived to reduce the success rate of the defender. From the literature survey, we ensure that none of the work addresses the problem of optimizing the number of system resources especially the number of APIs to be monitored while revealing a rootkit presence. Motivated by this, we propose a game theoretic model to detect rootkit presence by monitoring minimal number of APIs in windows operating system.

Reference Luo et al. (2010) discussed a non-cooperative, non-zero sum game which is played between the administrator and the attacker in a network of computers. The authors stated

that the game theoretic model enables the administrator to keep track of every action of the attacker so that preventing the attackers' near future attack is easy. In Hou et al. (2011), a resource optimization using game theoretic approach in uplink multi-cell Orthogonal Frequency Division Multiplexing (OFDMA) has been discussed. Their approach has two important steps. First, sub-carrier allocation in uplink has been done using integer programming. Secondly, power allocation has been distributed optimally using game theory model. Through simulation, the authors verified the successful integration of merits of distributed and centralized models. Barth et al. (2012) proposed a learning based reactive security strategy for defenders. The paper argued that the proposed reactive defenders strategies can outperform when proactive defenders protect attacks which can not actually happen. Today, many rootkit detectors detect malicious rootkit attacks by fetching and inspecting code and data sections of a page in memory. However, verifying/monitoring large code segment is very difficult and a time consuming process. To overcome this issue, the authors of Kinebuchi et al. (2013), presented limited local memory, a hardware-centric technique to monitor system integrity of a target operating system without being infected by malicious rootkit attacks. Their approach does not require the support of virtualization.

Few papers discussed the use of game theoretic model in cloud computing environment. Pilai and Rao (2014) proposed a game theory model for optimal resource allocation of cloud resources mechanism using coalition-formation and the uncertainty technique. Their approach efficiently allocates optimal resources to each computer to service more number of user requests on the cloud environment. In another paper Xu and Yu (2014), a game theoretic algorithm has been designed to get better cloud resource utilization rate. The authors achieved this goal by reducing the cloud resource disintegration when mapping a physical server to a virtual machine. The proposed resource allocation algorithm based on game theory achieved a higher resource utilization rate compared to existing resource allocation techniques.

## 3. Motivation

Intrusion Detection System (IDS) is defined as either a software or hardware that is used to monitor the activities occurring in a computer system (Host-based IDS) or network (Network-based IDS) in order to detect whether a malicious attack has occurred. Traditional HIDS monitors huge number of resources irrespective of the type of attack to be launched on the victim computer. We opt for a two-player repeated non-cooperative game model, since malicious code attacks are trying to compromise the victim computer repeatedly. Here, we specifically consider Native API hook attack which can be a part of malicious Rootkit Malware. We use a game-theoretic approach to optimize the resource consumption while detecting such kind of attacks. Our model dynamically selects a specific API targeted by Rootkit Malware based on the expected attack scenario.

### 3.1. Problem statement

The problem setup is twofold. First, we outline the game describing the players in the game theoretic model and the objective of the game which is played between two players. Next we devise strategies for the players.

### 3.2. Game approach

We assume that the game is played between the two players: the RRM and the RM. Here the RM is the attacker and the RRM is the defender. The objective of RM is to use '$n$' number of APIs (e.g. in worst case) from the victim computer with the intention of performing and launching some illegal activities. A malicious rootkit attack is successful when at least '$m$' APIs out of '$n$' APIs are utilized. It is assumed that a rootkit attack will not be achieved in a single step. So, we define our scenario as follows: The RRM detects the initial infection and predicts the next attack to be launched in the near future based on past experience or historical information. Then, the RRM immediately monitors additional APIs for '$t$' additional time period as the attack is expected to be launched.

### 3.3. Game strategies

To manipulate game strategies, we derive a game model for the interaction between the RRM and the RM. We consider (AS, CA, AR, T) where AS is the set of APIs equipped with RRM which we refer to as defender, CA is the system cost for monitoring additional API, AR is the set of attackers and $T = \{1, 2,...n\}$ is the set of target computers. To minimize the intermediate calculations, we select a two-player, non-cooperative game in which the number of repetition depends on the number of the attacking steps. Also we assume that both players know about the strategies and utility function they have.

The possible strategies for RRM are {no_attention, monitoring}. If the RRM detects a rootkit attack, it can select either to ignore the current task or to monitor. In case of monitor, the RRM will select more additional important APIs to monitor. The type of the API to be monitored and the length of the monitoring time highly depend on the information base of RRM. The monitoring time will be chosen from the information base, based on the attack scenario. When RRM detects initial rootkit attack which will be carried out in multiple steps, it will be able to detect the next possible attack action. At the same time, the RRM will choose to increase additional important APIs being monitored. After completing additional task, the RRM resumes monitoring the standard predefined number of APIs. On the other hand, the available strategies for the rootkit attacker i.e. RM are {end_process, proceed, waiting}. The strategy 'end_process' indicates to abandon the attack in order not to be detected; strategy 'proceeds' means proceeding with the predefined next step and strategy 'wait' indicates launching the next attack step after a certain period of time. Since predicting the delay time for every attack action is difficult, we will assume the delay time to derive the model.

## 4. Formulating the game

We assume that the RRM has been installed in a computer with a fresh copy of windows operating system. So the RRM will monitor only the important APIs in the system being monitored, and hence it is in a stable system resource cost. The set of APIs to be monitored by RRM is denoted as Ac = {a1, a2,...an}. The APIs are continuously monitored by RRM as

long as it is in live state. Now we denote $C_m$ as the additional computational system cost needed for monitoring single API at time '$t$', which is represented as:

$$C_m = r \times a$$

where '$a$' is a single API to be monitored and '$r$' is the number of clock pulses to monitor a single API and the time deviation is calculated using two local timestamps of the two events. Now the total increased system resource cost $\alpha$ is computed in case when the RRM chooses 'monitor' strategy. This can be represented as:

$$\begin{aligned}\alpha &= \text{Summation of drift in monitoring period of each additional API} \\ &= \text{Time taken to monitor all protected APIs} \\ &= \sum_{a \in A} C_m(t_m)\end{aligned}$$

where $t_m$ is the additional time to monitor single API. Then, converting these normal functions into utility function will be easy for us to apply game theory concept and it is given as:

$$R = f(\alpha(t_m, a)) = \gamma(\alpha(t_m, a))$$

where $\gamma$ is the weight parameter to describe the system cost of RRM. If RRM detects a rootkit attack, it will have utility gain of $\omega$, and $-\omega$ is the cost of damage to be caused. Hence the utility function of RRM with its possible outcomes is:

$$U_{\text{RRM}} = \begin{cases} \omega - C_m & \text{detect and stop} \\ -\beta\omega & t_m \leqslant t_d \\ \omega - C_m & t_m \geqslant t_d \text{ where } t_d > 0 \\ -\beta\omega - C_m & t_m < t_d \text{ where } t_m \geqslant 0 \end{cases}$$

where $t_m$ is the increased monitoring time because of the additional APIs to be monitored and $t_d$ is the delay in launching the near future attack or current attack and $\beta$ is false alarm rate of the RRM. On the other side, when the attack is successful, the attacker will gain $\omega'$ and $-\omega'$ otherwise. We define $C_a$ as the security APIs consumed by the attacker to make the next step to be successful. This is represented as:

$$C_a(t_d) = r \times a$$

where $r$ is the number of clock pulse to monitor a single additional API and $t_d$ is the delay time. Then, we define cost-Utility function for the attacker, which is given by:

$$R' = f(C_a(t_d)) = \gamma'(C_a(t_d))$$

where $\gamma'$ is the scale factor to define reward for a successful attack. Hence, the utility function of RM is given by:

$$U_{\text{RM}} = \begin{cases} 0 & \text{stop} \\ \omega' & t_m \leqslant t_d \\ \omega' - C_a & t_d \leqslant t_m \text{ where } t_m \geqslant 0 \\ -\omega' - C_a & t_d \leqslant t_m \text{ where } t_d > 0 \end{cases}$$

Table 1 shows the increased system resources for both the RRM and the RM in general format. The $S_i$ indicates different strategies to be selected by the RRM in which the monitoring time and the API being monitored will vary.

## 5. Evaluation

Since both players have the option of selecting a strategy randomly, we choose a mixed strategy to solve the game. In our game model, the expected cost of the RRM is computed as follows:

$$U_{\text{RRM}} = \sum_{i=1}^{m}\sum_{j=1}^{n} U_{\text{RRM}_{ij}} P_i Q_j$$

where $U_{RRM_{ij}}$ is the cost of RRM when it selects the strategy $i$ and the RM selects the strategy.

$j$ and $P_i$ is the probability of strategy $i$ for RRM, where $i = \{1, 2, \ldots, m\}$ and $m$ is the strategy options available, and $Q_j$ is the probability of strategy $j$ for RM, where $j = \{1\ 2, \ldots, n\}$ and $n$ is the strategy options available. As the players are rational, the RM always wants to minimize the expected payoff of the RRM. Therefore, RM will calculate the first derivative of URRM with respect to $P_i$. And RRM always wants to maximize the security level, i.e., by earning minimum payoff. So, we derive the probability for each strategy using maximin theorem.

### 5.1. Theorem 1. Selecting optimal strategy between two players

**Players:** {Rootkit Malware (RM $\leftarrow P^x$), Rootkit Revealing Module (RRM $\leftarrow P^y$)}
**Actions:** A ($P^x$) = {end_process (a1), proceed (a2), wait (a3)}

A ($P^y$) = {no_action (b1), monitor (b2)}

Then, the possible actions are: {(a1, b1), (a2, b2), (a3, b3)}.

The utility for $P^x$ is $U_x(a_i, b_j)$ and for $P^y$ is $U_y(a_i, b_j)$. A mixed strategy which uses a particular probability is represented as $P^x = \{p_1, p_2, ..., p_m\}$ where $p_i = P_r(a_i)$ is the probability for action $a_i$ to be played. Similarly, for the defender, $P^y = \{q_1, q_2, ..., q_n\}$. For each randomized strategy pair ($p$, $q$), the payoff $S(p, q)$ is represented as:

$$S(p, q) = \sum_{i=1}^{m}\sum_{j=1}^{n} p_i S(a_i, b_j) q_j$$

We employ $P$ and $Q$ to represent all mixed strategies available to $P^x$ and $P^y$. Since $P^y$'s objective is to select a randomized strategy $p$ from $P$ so as to maximize $S(p, q)$ i.e. maximize the minimum payoff. On the other side, $P^x$'s objective might be to select a randomized strategy $p$ from $Q$ to maximize its payoff which is equivalent to minimizing $S(p,q)$ i.e. minimize the expected payoff of $P^y$. For each mixed strategy $p$ which belongs to $P$, $P^y$'s security level is defined as: $v1(p) = \min_q S(p, q)$.

As $P^y$ wants to maximize the minimum payoff, it must choose strategy $p^*$ such that $v_1(p^*) \geqslant v_1(p) \forall_p \in P$. Let $v$ represents the maximum security level i.e. $v_1 = v_1(p^*)$.

$$\text{Then } v1(p) = \max_p v1(p) \geqslant v1(q) \text{ for all other mixed strategies.} \quad (1)$$

$$\text{Also, } v1 = \min_q S(p^*, q) \leqslant S(p^*, q) \forall_q \in Q \quad (2)$$

Eq. (1) says that the strategy that generates $v1$ is the best one in terms of maximizing security level. Eq. (2) implies that $v_1$ is the minimum payoff that $P^y$ can earn. If $P^x$ cannot play intelligently then $P^y$ will earn more than $v_1$. Similarly, if $P^y$ does not choose $p$ then $P^y$ can earn a lower payoff. The strategy $p^*$ is referred to be maximin strategy.

**Table 1** Utility derivation.

| Players | End_process | Proceed | Delaytime variation | |
|---|---|---|---|---|
| | | | $t_{d \leqslant t_m}$ | $t_d > t_m$ |
| No_attention | $\omega, 0$ | $-\beta\omega, -\omega'$ | $\omega - C_{mi}, -\omega' - C_{ai}$ | $-\beta\omega, \omega' - C_{ai}$ |
| $S_i(t_d, a_i)$ | $\omega - C_{mi}, 0$ | $\omega - C_{mi}, -\omega'$ | $\omega - C_{mi}, -\omega' - C_{ai}$ | $-\beta\omega - C_{mi}, \omega' - C_{ai}$ |

### 5.2. Pareto optimality

The monitor ($m$) strategy of RRM monitors an allocation of optimal number of APIs which cannot allow to improve the payoff of Rm. Specifically, we prove that strategy a is strongly pareto-optimal.

**Definition 1.** (*Pareto optimal*). Strategy $m$ is Pareto-Optimal, if there exist no strategy $m| = m$ such that $\forall_a Q_a(m|) > Q_a(m)$.

**Definition 2.** (*Strongly Pareto Optimal*). Strategy m is strongly Pareto-Optimal if there exist no strategy $m| = m$ such that $\forall_a Q_a(m|) \geqslant Q_a(m)$ and there exists $k$ transformations such that $Q_k(m|) > Q_k(m)$.

**Proposition 1.** Strategy $m$ is Pareto-Optimal.

**Proof.** For simplicity, let us consider $e = 1$. Since the strategy allocation vector for $S_1^{Final} = S_1^{First}$ there is no strategy that can produce a higher payoff. Also there exists no strategy $m| = m$Final can earn a higher payoff such that for all stages concurrently. Finally, to prove the strong Pareto-Optimal property, it is adequate to prove that a higher payoff can be achieved for RRM without affecting its optimal payoff at a any stage of $m$. □

**Theorem 2.** *The strategy m is strongly Pareto-Optimal.*

**Proof.** The proof is by induction on each stage subject of procession of API subsets $A_1 \supset A_2 \supset ..... \supset A_e$. First, we consider a single API, A1 = {1}, and likewise, as proven in the proof of proposition 1, detect that the consumption $S_1^{Final}$ cannot be changed. Let $e \geqslant 2$. Suppose the total number of APIs in the set Ae-1 cannot have their consumption modified in a manner to get better payoff. Now look at the subset of APIs $A_e = A_e \cup \{e\}$. APIs in each stage e can be partitioned into two disjoint subsets: $\hat{N}_e$ (the APIs in which the consumption is around to be set in eth stage or later) and $N_e/\hat{N}_e$ (the APIs in which the consumption is already fixed in eth stage). As the latter APIs are saturated, the increase in its APIs is ineffective as this cannot add to the boost of eth stage payoff (due to the API procession rate limit in stage Ae-1). However, if any change in passion of APIs, the payoff of some stage Ae-1 will also change, which contradicts the inductive statement. The remaining APIs are assigned in eth stage among each stage in $\hat{N}_e$ in a mode that gives up the largest payoff for stage e. Any variation from the strategy mFinal of the consumption of APIs in $\hat{N}_e$ would give lower payoffs i.e. optimal number of APIs. □

**Theorem 3.** A strategy choice $(p^*, q^*)$ in a game G is said to be an NE, if none of the players can increase its utility unilaterally diverging its strategy from it.

**Proof.** In short, neither the Rm nor the RRM can increase its utility unilaterally diverging from its current strategy. □

### 5.3. Case study

We do a case study to evaluate and simulate our game model. Since delay time in launching near future attack will vary from attack to attack, let's set it from the historical data. We collected 100 different rootkit samples which were obtained from http://www.offensivecomputing.net which adopt both user-mode hook and kernel-mode hook and analyzed them in windows XP virtual machine. By observation, most of the rootkit samples affect common Native API functions to perform illegal activities in the victim computer. The RRM's information base contains most commonly affected Native API functions and their respective Dynamic Link Libraries (.dll) file. Our evaluation describes the game after the initial attack action; in our case it is infecting all executable _les in the victim computer which is being detected by the RRM. Now, the RRM will take narrow action based on its intelligence about the impending attack following the infecting executables in the attack scenario i.e. from (no attention, increasing the additional APIs to be monitored by 50, and monitoring time duration by 400 s, 900 s, and 1500 s). Also, we choose $\gamma = 2$, $d = 0.020$, $-\omega = 3500$, and $-\omega' = 2500$. Table 2 shows the manipulated results for the first round.

As we mentioned earlier, the players are rational and also the attacker knows defender strategies and will try to maximize its gain. The optimized result for $P^x$ and $P^y$ is calculated using Gambit tool. Tables 3 and 4 contain the game result for Table 2.

**Table 2** General IAT hook.

| | End_process | Proceed | 800 | 1200 |
|---|---|---|---|---|
| No_attention | 3500, 0 | −3500, 2000 | −3500, 900 | −3500, 300 |
| 400 | 2700, 0 | 2700, −3000 | −4300, 900 | −4300, 300 |
| 900 | 1100, 0 | 1100, −3000 | 1100, −4100 | −5900, 300 |
| 1500 | 500, 0 | 500, −3000 | 500, −4100 | 500, 4700 |

Finally, we form the payoff matrix for general Rootkit Malware detection with four different strategies as shown in Table 5.

For simplicity we used binary values to represent the output from rootkit detection module i.e. value 1 represents success and value 0 represents failure.

## 6. Simulation result

We have selectively taken 10 different rootkit samples which adopt hooking technique from running processes to build RRM's information base. The samples are different from each other in terms of dll's, APIs and their corresponding functions. For every scenario, we have simulated a game theory model to calculate payoff matrix. From the calculated payoff matrix, the Nash Equilibrium is generated using Gambit tool. Each scenario is simulated 10 times in our approach to calculate its corresponding utility resource. The average of every scenario is found and is plotted in a graph using MatLab software. In addition, the same 10 different samples are simulated in traditional approach and corresponding resource values are calculated to discover the difference. Fig. 1 shows that our model takes less resource consumption than traditional approach.

Fig. 2 depicts the detection accuracy of our model. The *X*-axis illustrates different scenarios with different Rootkit Malware samples. The *Y*-axis illustrates the detection rate. We believe that the traditional Intrusion Detection System's accuracy rate is 95% (Liu et al., 2008). From Fig. 2, we guarantee that our model's detection accuracy is similar to traditional detection accuracy with small difference (1%). This is because, our system does not focus to detect all types of malware on the victim computer. Instead, our system runs at random interval to detect Rootkit Malware attacks by monitoring Native API hooks.

Bearing the above two graphs in mind, we make sure that our approach can greatly save system resource consumption without losing traditional detection accuracy. As the need for increased computing resource to achieve better performance, our model is well suited for a host which runs under limited resource.

### 6.1. One attacker and one defender

Though multiple malware attacks can be tried on one victim computer, we concentrate only to optimize resource consumption of RRM during malicious rootkit attacks which use Native API hook technique. We simulate and model malicious rootkit attack with one attacker (RM) and one defender (RRM). The RM resource $P$ and the RRM resource $Q$ are both initially set to 1. Assume that both RM and RRM focus on the sensible Native APIs for example six number of APIs. When $x^* = 1$, $y^* = 1$ then $p_i^* = 0.00074$, $q_i^* = 0.447$, Nmin = 6. If we calculate (URM)max denotes the maximum payoff of the RRM and $(U_{RRM}^*)_{\min}$ denotes the minimum payoff of the RRM when it works on $q^*$ and RM selects its strategy to maximize its payoff. The simulated results show that the NE has the optimal resource consumption for the RRM when RM intelligently chooses its strategy to increase its payoff.

**Table 3** Optimal payoff for $P^x$.

| Payoff | No_attention | 20 | 30 | 50 |
|---|---|---|---|---|
| 300 | ½ | 4/35 | 0 | 27/70 |

**Table 4** Optimal payoff for $P^y$.

| Payoff | End_process | Proceed | 30 | 50 |
|---|---|---|---|---|
| 0 | 13/20 | 7/50 | 0 | 21/100 |

**Table 5** General rootkit detection payoff.

| | RegistryKey | IAT | Inline | SSDT |
|---|---|---|---|---|
| No_attention | 1, 0 | 0, 1 | 0, 1 | 0, 1 |
| IAT | 1, 0 | 1, 0 | 0, 1 | 0, 1 |
| IAT, inline | 1, 0 | 1, 0 | 1, 0 | 0, 1 |
| IAT, inline, SSDT | 1, 0 | 1, 0 | 1, 0 | 1, 0 |

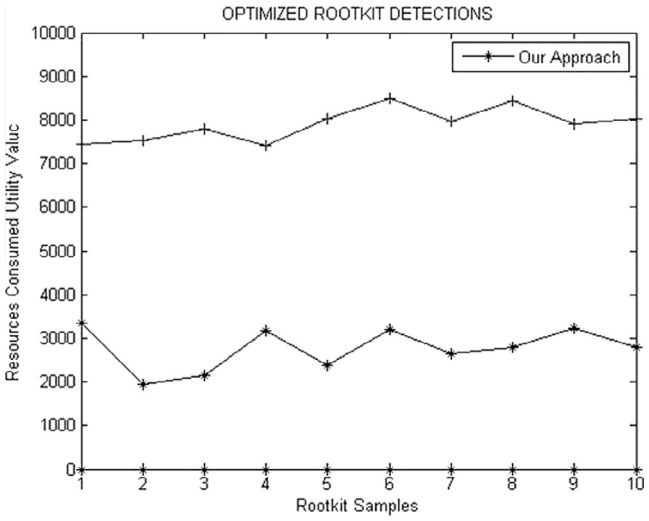


**Figure 1** Resource consumption.

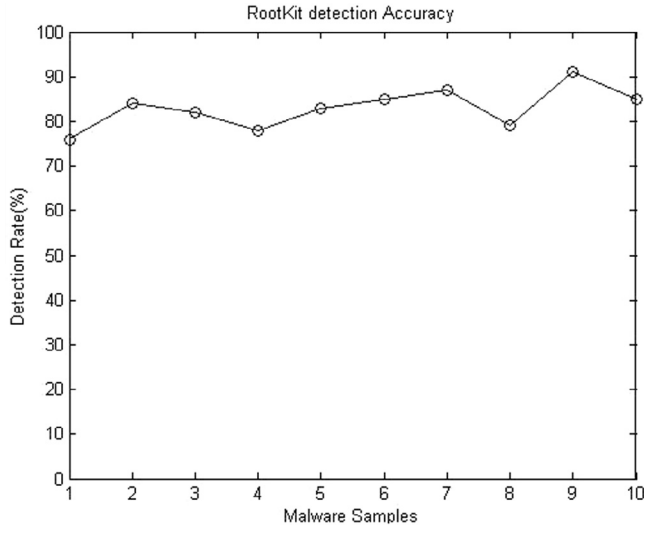


**Figure 2** Detection accuracy.

The dynamic game model can be visualized as follows. Assume that the attacker and the defender game can start at the susceptible state. The game ends when RM selects end_process or the RRM reacts to the attack. Similarly, if RM succeeds and the status shifted to a new susceptible state, which is not a game element, the game ends. On the other hand, if RM succeeds and the status shifted to any one of the game factors, the game carries on with a new start. Finding a NE will yield solution to this game.

### 6.2. Metaheuristic optimization

Metaheuristic Optimization (MO) solves many real world optimization (either minimization or maximization) problems using metaheuristic algorithms. As resources are limited, especially while detecting malicious rootkit code attacks, the optimal consumption of these resources is essentially important. Though this paper improves the detection accuracy of the proposed system using game theoretic approach when compared to traditional approaches, the resources consumed by RRM still can be optimized using MO algorithms such as genetic algorithms, Bee algorithms and Ant Colony optimization.

## 7. Conclusion

For analysis, we have taken traditional rootkit detection technique which acquires huge computing resources to monitor IAT, INLINE, and SSDT data structures in the end-system. We have analyzed the traditional approach and identified that the number of APIs monitored in these data structures is quite high. But the rootkit developers write their code to hook most common dll's and their functions. In order to reduce the number of monitoring APIs, we have analyzed various rootkit samples and found common APIs and dlls which are mostly affected by malware rootkits. We have taken two-player, non-zerosum, and non-cooperative game model. And we have simulated the game between rootkit and our approach to find the best strategy for both players. The simulation results show that our model acquires less system resources than the traditional rootkit detection technique without losing detection accuracy. Our game model is a suitable choice where the system resource is a critical component.

## References


Abdullah Al-Kadhi, Mishaal, 2011. Assessment of the status of spam in the Kingdom of Saudi Arabia. J. King Saud Univ. – Comput. Inform. Sci. 23, 45–58.

Alfantookh, Abdulkader A., 2006. DoS attacks intelligent detection using neural networks. J. King Saud Univ. – Comput. Inform. Sci. 2, 27–45.

Alpcan, T., Baskar, T., 2004. A game theoretic analysis of intrusion detection in access control systems. In: Proc. of the 43rd IEEE Conf. Decision and control (CDC). Bahamas.

Altwaijry, Hesham, Algarny, Saeed, 2012. Bayesian based Intrusion Detection System. J. King Saud Univ. – Comput. Inform. Sci. 24, 1–6.

Barth, A., Rubinstein, Benjamin I.P., Sundararajan, Mukund, 2012. A learning-based approach to reactive security. IEEE Trans. Depend. Secure Comput. 9 (4), 1–2.

Chen, Lin, Leneutre, Jean, 2009. A game theoretical framework on intrusion detection in heterogeneous networks. IEEE Trans. Inform. Forensics Security 4 (2), 165–178.

Emigh, A., 2006. The crimeware landscape: malware, phishing, identity theft and beyond. J. Digital Forensic Pract. 1 (3), 245–260.

Game theory. Available: http://en.wikipedia.org/wiki/Gametheory.

Hou, Zhao, Cai, Yueming, Dan, Wu, 2011. Resource allocation based on integer programming and game theory in uplink multi-cell cooperative OFDM systems. Eurasip J. Wireless Commun. Network, 1–10.

Kinebuchi, Yuki, Butt, Shakeel, Ganapathy, Vinod, Iftode, Liviu, Nakajima, Tatsuo, 2013. Monitoring integrity using limited local memory. IEEE Trans. Inform. Forensics Security 8 (7).

Kodialam, M., Lakshman, T.V., 2003. Detecting network intrusions via sampling: A game theoretic approach, In: IEEE INFOCOM, San Franciso.

Liu, P. A game theoretic approach to cyber attack prediction. DOE ECPI Program Final Technical Report, 2005.

Liu, Y., Man, H., Commaniciu, C., 2002. A game theoretic approach to efficient mixed strategies for intrusion detection. IEEE Int. Conf. Commun. (5), 2201–2206

Liu, Y., Comanciciu, C., Man, H., 2006. Modeling misbehavior in adhoc networks: a game theoretic approach for intrusion detection. Int. J. Network Security 1 (1), 243–254.

Shuai Liu, Da Yong Zhang, Xiao Chu, Hadi Otrok, Prabir Bhatacharya, 2008. A Game theoretic Approach to Optimize the Performance of Host-based IDS. In: IEEE Int. Conf. on wireless and Mobile Computing, Networking and Communication. pp. 448–453.

Shuai Liu, Da Yong Zhang, Xiao Chu, Hadi Otrok, Prabir Bhattachrya, 2008. A game theoretic approach to optimize the performance of host-based IDS. In: Proc. of the IEEE International Conference on Wireless & Mobile Computing, Networking & Communication. pp. 448–453.

Luo, Yi, Szidarovszky, Ferenc, Al-Nashif, Youssif, Hariri, Salim, 2010. Game theory based network security. J. Inform. Security, 41–44.

Oskar, Von Neumann J., Oskar, M., 2004. Theory of Games and Economic behavior. Princeton University Press.

Otrok, H., Mehrandish, M., Assi, C., Debbabi, M., Bhatacharya, P., 2008. Game theoretic models for detecting network intrusions. Comput. Commun. 31 (10), 1934–1944.

Pacha, A., Park, J.M., 2006. A Game theoretic formulation for intrusion detection in mobile adhoc networks. Int. J. Network Security 2 (2), 146–152.

Pilai, Parvathy S., Rao, Shrisha, 2014. Resource allocation in cloud computing using the uncertainty principle of game theory. IEEE Syst. J. 1, 1–12.

Quynh, N.A., Take Fuji, Y., 2007. Towards a tamper-resistant kernel rootkit detector. In: Proc. of the 2007 ACM symposium on Applied Computing, South Korea. pp. 276–283.

W3Schools, OS platform statistics, Available: http://www.w3schools.com.

Wang, L., Dasgupta, P., 2007. Kernel and application integrity assurance. Ensuring freedom from Rootkits and Malware in a computer system. In: Proc. of the 21st Int. Conference on Advanced Information Networking and Application Workshops. IEEE Computer Society.

Xu, Xin, Yu, Huiqun, 2014. A game theory approach to fair and efficient resource allocation in cloud computing. Math. Prob. Eng. 2014, 1–14.