Probabilistic Inference on Integrity for Access Behavior Based Malware Detection

Weixuan Mao[1], Zhongmin Cai[1](✉), Don Towsley[2], and Xiaohong Guan[1]

[1] MOE KLINNS Lab, Xi'an Jiaotong University, Xi'an, Shaanxi, China
{wxmao,zmcai,xhguan}@sei.xjtu.edu.cn

[2] School of Computer Science, University of Massachusetts, Amherst, MA, USA
towsley@cs.umass.edu

Abstract. Integrity protection has proven an effective way of malware detection and defense. Determining the integrity of subjects (programs) and objects (files and registries) plays a fundamental role in integrity protection. However, the large numbers of subjects and objects, and intricate behaviors place burdens on revealing their integrities either manually or by a set of rules. In this paper, we propose a probabilistic model of integrity in modern operating system. Our model builds on two primary security policies, "no read down" and "no write up", which make connections between observed access behaviors and the inherent integrity ordering between pairs of subjects and objects. We employ a message passing based inference to determine the integrity of subjects and objects under a probabilistic graphical model. Furthermore, by leveraging a statistical classifier, we build an integrity based access behavior model for malware detection. Extensive experimental results on a real-world dataset demonstrate that our model is capable of detecting 7,257 malware samples from 27,840 benign processes at 99.88 % true positive rate under 0.1 % false positive rate. These results indicate the feasibility of our probabilistic integrity model.

Keywords: Probabilistic graphical model · Integrity protection · Malware

1 Introduction

In spite of considerate effort by security researchers and engineers, attackers continue to craft malicious code (malware). A recent security threat report by Symantec states that there were more than 317 million new pieces of malware created in 2014, which is 26 % more than in 2013 [27]. Being faced with ever-growing and increasingly sophisticated malware, it is important to develop more effective defenses from essential perspectives of security [28].

Integrity protection has proven an effective way of malware detection and defense [4,7,23,28]. Determining the integrity of subjects and objects is fundamental to integrity protection [26]. The integrity of a subject or an object refers to the trustworthiness of its contents. Integrity is typically divided into levels,

H. Bos et al. (Eds.): RAID 2015, LNCS 9404, pp. 155–176, 2015.
DOI: 10.1007/978-3-319-26362-5_8

e.g., *trusted* and *untrusted*, *high* and *low*, etc. Integrity protection aims at preventing trusted objects from being accessed inappropriately, e.g., it forbids an untrusted program from writing a trusted file. Prior work manually defines the number of integrity levels, and assigns integrity levels to subjects and objects either manually or by rules. For example, previous work usually defines two integrity levels [4,7,26]. Learning from previous attacks, Sun et al. treat Media players and games as having low integrity [26], LOMAC treats system files as having high integrity, and builds on a rule that assigns a subject a low integrity, if it read an object with low integrity [7]. Windows Vista treats Internet Explorer as having low integrity, and Vista files as having high integrity [11]. It is feasible to define integrity levels either manually or by rules, when the number of subjects and objects is small, and the behaviors of subjects are simple. However, given hundreds of thousands subjects and objects in a modern operating system, it is challenging and error-prone to assign integrity levels to all of them manually. And, given the intricate behaviors of subjects, it is neither flexible nor adaptive to assign integrity levels to them by rules.

This paper aims to determine integrity levels for all system subjects, i.e., programs, and system objects, i.e., files and registries, based on access behaviors of benign programs. Our method builds on two primary security policies, "*no read down*" and "*no write up*". These two policies, first proposed by Biba [4], have become the basis of integrity protections in modern operating systems [7,12,26]. These policies make connections between integrity levels and access behaviors. Our method determines the integrity levels of subjects and objects based on these connections. Unlike earlier work, it makes no assumption of the total number of integrity levels and needs no knowledge obtained from previous attacks.

The integrity level of system subjects and objects may change in different executions and contexts. We model integrity levels as random variables to capture their uncertainties. Meanwhile, each observation of access behavior, involving a pair of system subject and object, implies an ordering of their integrity levels between them, according to "no read down" and "no write up" security policies. This ordering of integrity level defines a joint integrity level (joint distribution) of the subject-object pair. Thus, we build a probabilistic generative model to capture these connections, and derive the joint distribution of the integrity levels of each pair, based on observations of access behaviors of benign programs. To obtain the integrity levels of each subject and object, we aggregate the joint integrity level of each pair to build a pairwise Markov network. We then employ message passing based inference on this pairwise Markov network, i.e., loopy belief propagation, to obtain integrity levels of all subjects and objects.

Furthermore, we employ a statistical classifier to build an integrity based access behavior model for malware detection. We conduct a set of experiments to evaluate our model by comparing with baseline models, on a data set consisting of 27,840 executions of 534 benign programs and 7,257 executions of malicious programs. Our encouraging results demonstrate the usefulness of our access behavior model for malware detection based on determined integrity levels, and the feasibility of our model on determining integrity levels for system subjects and objects.

The contributions of this paper are summarized as follows:

– Modeling integrity level of system subjects and objects under two primary security policies. We propose a probabilistic generative model to capture the connection between the joint integrity level of each pair of system subjects and objects, and observations of access behaviors of benign programs, corresponding to "no read down"and "no write up" security policies.
– Probabilistic graphical model on joint integrity levels of all subject-object pairs. We build a probabilistic graphical model, pairwise Markov network, to characterize joint integrity levels of all pairs, and leverage a message passing based inference to further characterize integrity levels of all subjects and objects.
– An integrity based access behavior model for malware detection. We employ a statistical classifier, random forest, to build an access behavior model for malware detection, based on integrity levels of accessed objects of programs.
– Extensive experimental results on a real data set demonstrate the feasibility and capability of our model. On a data set consisting of 27,840 executions of 534 benign programs and 7,257 executions of malicious programs, our experimental results of malware detection exhibit a 99.88 % true positive rate at 0.1 % false positive rate.

The paper is organized as follows. We first introduce the background and related work in Sect. 2. We then explain our probabilistic integrity model and inference, and our model of malware detection in Sect. 3. After demonstrating our experimental results in Sect. 4, Sect. 5 concludes our work and states future work.

2 Background and Related Work

2.1 Integrity Protection

Integrity protection has been demonstrated to be an effective way of protecting modern operating systems [7,12,23,26]. It determines when information flows are allowed based on the integrity levels of subjects and objects. One of the most fundamental and well-known integrity protection models is the Biba model [4], which defines policies as follows.

– A subject is allowed to read an object only if its integrity is lower than or equal to that of the object (no read down).
– A subject is allowed to write an object only if its integrity is higher than or equal to that of the object (no write up).

The Biba model provides security policies for integrity protection, and was initially proposed as a mandatory access control (MAC) model in military, government systems. We refer to "no read down" as *NRD*, and "no write up" as

NWU in the following paper. However, Biba is too restrictive for modern operating systems [21]. Follow-up research aimed to build a more applicable model from two perspectives: (1) Determine the integrity level of subjects and objects. (2) Devise policies for these integrity levels.

Prior work assigns integrity levels to subjects and objects, and devises policies under integrity levels either manually or by rules [7,23,26]. LOMAC is a Linux extension of Biba model. It defines two integrity levels, high and low. System files are assigned high integrity levels and the network is assigned a low integrity level. LOMAC changes the integrity level of subjects and objects according to the *low water mark* principle, i.e., if a subject with high integrity reads an object with low integrity, the integrity of the subject is assigned to low [7]. This provides a mechanism of capturing changes of integrity levels of subjects and objects dynamically. However, its way of determining static integrity levels of subjects and objects relies on limited rules. Sun et al. devise elaborate rules to assign integrity levels, either high or low, to subjects and objects, and define policies based on their experiences on benign and malicious information flow [26]. In their later work, they apply integrity protection with similar policies into software installation, and propose a Secure Software Installer to prevent untrusted software from modifying files used by trusted software [25]. Their rules require a large number of examples of both attacks and defenses. Although these rules may provide an effective protection, it is expensive to devise such rules, and difficult to make them flexible enough to deal with newly emerging attacks and benign programs. Hsu et al. provide a framework for recovering from malicious programs and their effects based on integrity. They focus on the NRD policy, because they do not aim to prevent malware but only to recover from intrusions [12]. Mao et al. leverage graph centrality to measure the importance of systems subjects and objects, which is treated as a proxy for integrity [20]. However, there is still a gap between the importance of graph structure and integrity in operating systems. Beside these efforts by security researchers, one commercial example is the Mandatory Integrity Control (MIC) in Windows Vista. It labels the integrity levels of file objects as either Low, Medium, High, or System. As a default, critical Vista files are labeled System, other objects are labeled Medium, and Internet Explorer is labeled Low. MIC designs policies to enforce security with the help of user account control (UAC) [1,11,19]. However, it highly relies on judgment of users, which makes it unusable for ordinary users [3].

Previous integrity protection model suffer from two shortcomings. Firstly, it is difficult to determine the integrity levels of all subjects and objects, as there are hundreds of thousands and intricate behaviors between them. Prior work usually manually assigns integrity levels to a small subset of subjects and objects based on their experiences in attacks, and then uses a rule-based approach to determine others [7,11,25,26]. Unlike prior work, which predefines the number of possible integrity levels, we make no assumption about the number of integrity levels in operating systems, but determine it from access behaviors of benign programs. We aim to determine the integrity level for each subject and object, relying on knowledge of benign programs, but not malicious programs. Secondly, mandatory control policies and manually devised rules cannot accommodate real-world operation very well without the introduction of numerous exceptions in the face of crafty attackers and diverse benign programs. We employ a statistical classifier to extract policies for benign and malicious programs with respect to their access behaviors under our integrity levels.

2.2 Behavior Based Malware Detection

Prior work on behavior based malware detection usually employs heuristic, rule-based or statistical learning algorithms to construct behavior models or specifications for programs [2,6,8,16,17]. It successfully trades false positives off against false negatives. However, it relies on statistical discriminations between benign and malicious programs only, and neglects the essence of malware from a security perspective, i.e., violations of security policies. Our method leverages the integrity level of objects involved in access behaviors, which pays more attention to violations of security policies of programs. Under the integrity level derived from our model, we observe discriminations not only in statistics, but also from a security perspective in our experimental results.

3 Methodology

3.1 Overview

In this section, we present our method for deriving integrity levels of system subjects and objects, and malware detection. It consists of two components: (1) Probabilistic integrity model and inference. (2) Malware detection utilizing the integrity levels. Figure 1 illustrates the framework of our method in this paper.

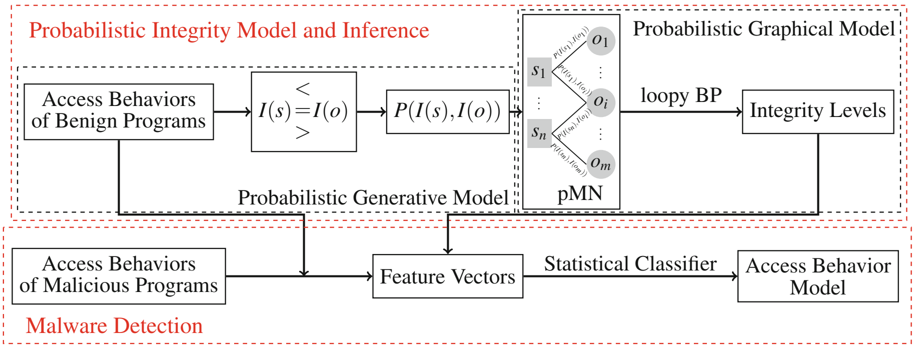

Probabilistic Integrity Model and Inference
Access Behaviors of Benign Programs
$I(s) \overset{<}{\underset{>}{=}} I(o)$
$P(I(s),I(o))$
Probabilistic Generative Model
s_1
s_n
o_1
o_i
o_m
pMN
Probabilistic Graphical Model
loopy BP
Integrity Levels
Access Behaviors of Malicious Programs
Feature Vectors
Statistical Classifier
Access Behavior Model
Malware Detection

Fig. 1. Framework of our method for deriving integrity levels and malware detection

Since the integrity levels of system subjects (programs) and objects (files or registries) may change in different executions and contexts, we represent the integrity levels as random variables to capture their uncertainties in real operating systems. The first component of our method, probabilistic integrity model and inference, consists of two steps: (i) Given all observed access behaviors, we infer the joint integrity level (joint distribution) of each subject-object pair, under proposed a probabilistic generative model for access behaviors. Depending on which of the two primary security policies NRD and NWU applies, each observed access behavior implies an ordering of the integrity levels of a subject-object pair. The possible orderings include $<$, $=$, $>$. Moreover, the orderings within a subject-object pair implies its joint integrity level (joint distribution). (ii) We aggregate the joint distributions of integrity levels for all pairs of subjects and objects to construct a pairwise Markov network (pMN) that provides a joint distribution of integrity levels for all subjects and objects. Our goal is to characterize the distribution of the integrity levels of each subject and object from the pMN. Furthermore, we employ loopy belief propagation (loopy BP), a message passing approximate inference algorithm, to infer the marginal distributions of the integrity levels for subjects and objects.

Once we characterize the integrity levels, we extract feature vectors to describe access behaviors of programs. Resorting to a statistical classifier, we build a model of malware detection, that accounts for the security meanings of access behaviors. In this paper, we focus on system subjects and objects, i.e., programs, files and registries.

3.2 Probabilistic Integrity Model

Probabilistic Integrity and Pairwise Relationship. It is not easy to determine the exact integrity level of subjects and objects. For example, due to the varying behaviors and intents, a program may exhibit different integrity levels in different executions and contexts. Meanwhile, accessed by these programs, the integrity level of a file or registry may be changed according to security policies, e.g., low/high water mark principle [7], etc. We capture these uncertainties by modeling the integrity level of subjects and objects as random variables, and make two primary assumptions in this paper.

Primary assumptions: (1) We assume the integrity level of each subject or object i is a random variable $I(i) \in \{L, H\}$, where L and H indicate low and high integrity levels. (2) We assume observed access behaviors of benign programs obey security policies NRD and NWU to ensure a secure operating system.

For each subject-object pair (s, o), we denote their joint integrity level as a random variable $J_I(s, o)$, where $P(J_I(s, o) = (x, y)) = P(I(s) = x, I(o) = y)$, $x, y \in \{L, H\}$. Meanwhile, given a pair of integrity levels (x, y), we say that $x < y$ if $x = L$ and $y = H$. Hence, there are three possible orderings between a pair of s and o considering their integrity levels, i.e., $I(s) < I(o)$, $I(s) = I(o)$, $I(s) > I(o)$. Note that, the integrity level of s may be higher than that of o in one execution, but lower than that of o in another execution. Therefore, we represent the order between s and o as a random variable $E_I(s, o)$ with three possible values, i.e., $E_I(s, o) \in \{I(s) < I(o), I(s) = I(o), I(s) > I(o)\}$. Thus, we obtain $P(J_I|E_I)$ by $P(J_I|E_I) = \frac{P(E_I|J_I)P(J_I)}{P(E_I)} \propto P(E_I|J_I)P(J_I)$ as follows.

(1) If E_I is $I(s) < I(o)$, then,

$$P(J_I = (L,H)|E_I) = 1; \text{ otherwise, } 0. \tag{1}$$

(2) If E_I is $I(s) = I(o)$, then,

$$P(J_I = (L,L)|E_I) = P(LL), P(J_I = (H,H)|E_I) = P(HH); \text{ otherwise, } 0, \tag{2}$$

where $P(LL) = \frac{P(J_I=(L,L))}{P(J_I=(L,L))+P(J_I=(H,H))}$, $P(HH) = \frac{P(J_I=(H,H))}{P(J_I=(L,L))+P(J_I=(H,H))}$

(3) If E_I is $I(s) > I(o)$, then,

$$P(J_I = (H,L)|E_I) = 1; \text{ otherwise, } 0. \tag{3}$$

Probabilistic Generative Model for Access Behavior. An access event is an observed access behavior. We refer to system call events related to files or registries as *access events*. Each access event involves a subject and an object, and we divide all access events into two types, *read* and *write*, according to their information flows [13].

One execution of a program s consists of a set of access events. In each execution, there are three possible access behaviors between a subject s and an object o, i.e., *Read-only* (r), *Write-only* (w), *Read* & *Write* (r & w) [13]. Thus, once s accesses o, we represent the access behavior of s on o as a random variable with three possible values, and s takes one of three possible access behaviors on o in each execution. We denote the probabilities that s reads-only, writes-only, and reads & writes o as $P_r(s,o)$, $P_w(s,o)$, and $P_{r\&w}(s,o)$.

Furthermore, for a program s with different executions, we assume the access behavior of s in one execution is independent of its behavior in other executions. We define $Acc(s,o)$ to be the access behavior of s on o among all executions, consisting of $N_{r,(s,o)}$, $N_{w,(s,o)}$ and $N_{r\&w,(s,o)}$, which are the number of executions that s reads-only, writes-only and reads & writes, and they obey a multinomial distribution, i.e., $Acc(s,o) = (N_{r,(s,o)}, N_{w,(s,o)}, N_{r\&w,(s,o)}) \sim \text{Multi}(N_{(s,o)}, P_r(s,o), P_w(s,o), P_{r\&w}(s,o))$, where $N_{(s,o)}$ is the total number of executions of program s that accesses object o.

With our second primary assumption, the relationship between the access behavior of s on o and their integrity levels can be interpreted as follows: If $I(s) < I(o)$, then s reads-only o, i.e., $P_r(s,o) = 1$ and $P_w(s,o), P_{r\&w}(s,o) = 0$; if $I(s) > I(o)$, then s writes-only o, i.e., $P_w(s,o) = 1$ and $P_r(s,o), P_{r\&w}(s,o) = 0$; if $I(s) = I(o)$, then s can perform any of behaviors, i.e., $0 < P_r(s,o), P_w(s,o), P_{r\&w}(s,o) < 1$. However, as mandatory security policies, there may exist violations of them in commercial operating systems [21]. Our model allows violations of security policies by assuming the distribution of access behavior of s on o not only depends on the order of integrity levels between them, but also on the distribution of access behavior under violations. We assume the distribution of access behavior under violations is identical to that under equal integrity, i.e., $I(s) = I(o)$. Thus, the distribution of all access behaviors is derived by combining that under order of integrity levels and that under violations, which results in a random variable T representing the prior distribution of the distribution of access behavior. Based on the above analysis and assumptions, we build a probabilistic generative model, which is a hierarchical Bayesian model, for the access behavior of programs and accessed objects with their integrity levels as shown in Fig. 2. Assumptions of conditional probabilities in the model are presented in Eqs. (4)–(7).

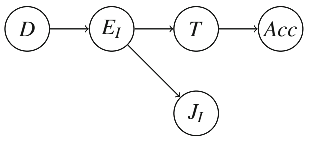


Fig. 2. A generative model for the access behavior with integrity levels

Equation (4) presents a Dirichlet prior D of the integrity ordering to deal with the lack of a sufficient number of observations in the data set. Equation (5) indicates the categorical distribution of the integrity ordering, where the probability of $I(s) < I(o)$, $I(s) = I(o)$, and $I(s) > I(o)$ are d_1, d_2, and d_3 respectively. Before presenting the multinomial distribution of access behaviors with parameters t_1, t_2, and t_3 in Eq. (7), Eq. (6) presents the prior T of this multinomial distribution conditioning on the integrity ordering, which aims to model access behaviors by combining both the security policies, i.e., NRD and NWU, and potential violations in commercial operating systems. Combining with the relationship between E_I and J_I as shown in Eqs. (1)–(3), this generative model provides a way to model the access behavior with joint integrity level of subjects and objects. Moreover, it offers a way to infer the joint integrity level $J_I(s, o)$ given observations of access behaviors, which is presented in the following subsection.

$$D = (d_1, d_2, d_3) \sim \mathrm{Dir}(\alpha_1, \alpha_2, \alpha_3), \tag{4}$$

$$E_I | D \sim \mathrm{Cat}(3, d_1, d_2, d_3), \tag{5}$$

$$T = (t_1, t_2, t_3) | E_I \sim \begin{cases} \mathrm{Dir}(1 + \beta_1, \beta_2, \beta_3), & \text{if } I(s) < I(o), \\ \mathrm{Dir}(\beta_1, \beta_2, \beta_3), & \text{if } I(s) = I(o), \\ \mathrm{Dir}(\beta_1, 1 + \beta_2, \beta_3), & \text{if } I(s) > I(o), \end{cases} \tag{6}$$

$$Acc | T \sim \mathrm{Multi}(N, t_1, t_2, t_3). \tag{7}$$

Probabilistic Graphical Model on Integrity. Our goal is to characterize the integrity level of each subject and object by marginal distributions, i.e., $P(I(s))$ and $P(I(o))$. To achieve this goal, we need to aggregate the joint integrity levels of all pairs of subjects and objects, and calculate the marginal integrity distribution for subjects and objects. We achieve this goal using a probabilistic graphical model, or more accurately, *pairwise Markov network* (pMN). Pairwise Markov networks are the simplest subclass of Markov networks. A pMN is an undirected probabilistic graphical model $G(V, E, \Psi)$, where V is a set of nodes representing random variables, E is a set of edges representing relationships between nodes with factors defined in Ψ, each edge is associated with a factor over a pair of nodes [14]. In our problem, the pMN is a bipartite graph, where each node represents the integrity level of a subject or object, a subject is connected with an object if there exists an observed access event associated with them. There is no edge between two subjects or two objects. Meanwhile, we encode the joint integrity level J_I for each pair of subject and object into each edge, the factor on each edge is joint integrity level J_I. Figure 3 illustrates an example of a pMN consisting of four pairs of subjects and objects. Since the bipartite graph constructed from a real data set contains loops, we compute an approximate inference on the integrity level of each subject and object as shown in the following subsection.

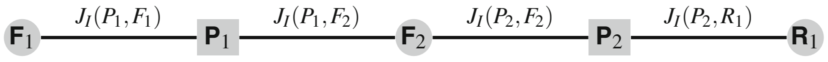


Fig. 3. An example of pairwise Markov network

3.3 Probabilistic Inference

Inference on Joint Integrity Level of Each Pair. Under the generative model shown in Fig. 2, the Bayes estimator $\hat{P}_E$ for the integrity ordering given access events is

$$\begin{aligned}\hat{P}_E = P(E_I|Acc) &= \sum_{D,T} P(E_I, D, T|Acc), \propto \sum_{D,T} P(Acc|T)P(T|E_I)P(E_I|D)P(D),\\ &= \sum_{T} P(Acc|T)P(T|E_I) \sum_{D} P(E_I|D)P(D). \end{aligned} \tag{8}$$

More specifically, the probabilities of all possible orderings are

$$\hat{P}_{E_I}(<) = P(<|Acc) = \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3}\frac{(\beta_1+\beta_2+\beta_3)(N_r+\beta_1)}{\beta_1(N+\beta_1+\beta_2+\beta_3)}/\Sigma, \tag{9}$$

$$\hat{P}_{E_I}(=) = P(=|Acc) = \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3}/\Sigma, \tag{10}$$

$$\hat{P}_{E_I}(>) = P(>|Acc) = \frac{\alpha_3}{\alpha_1+\alpha_2+\alpha_3}\frac{(\beta_1+\beta_2+\beta_3)(N_w+\beta_2)}{\beta_2(N+\beta_1+\beta_2+\beta_3)}/\Sigma, \tag{11}$$

where $\Sigma = \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3}\frac{(\beta_1+\beta_2+\beta_3)(N_r+\beta_1)}{\beta_1(N+\beta_1+\beta_2+\beta_3)} + \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3} + \frac{\alpha_3}{\alpha_1+\alpha_2+\alpha_3}\frac{(\beta_1+\beta_2+\beta_3)(N_w+\beta_2)}{\beta_2(N+\beta_1+\beta_2+\beta_3)}$. More details of the derivation are presented in Appendix.

Our estimator $\hat{P}(J_I(s,o))$ for the joint distribution of integrity levels is,

$$\hat{P}(J_I(s,o)) = P(J_I(s,o)|Acc) = \sum_{E_I(s,o)} P(J_I(s,o)|E_I(s,o))P(E_I(s,o)|Acc). \tag{12}$$

Inference on Integrity Level of Subject and Object. There exist loops in our pairwise Markov network. Hence, the estimation of marginal distributions for such graphs is known to be NP-complete. Loopy belief propagation provides an approximate and efficient way of inference based on *message passing*. It has proven to be a successful at inferring on marginal distributions over loopy graph in various domains, such as object tracking in computer vision, error-correcting code, etc [14]. In particular, researchers have applied loopy belief propagation to solve problems in security area by modeling them as classification problems [18, 29]. We apply this method to our problem to infer the probabilistic integrity level of each subject and object.

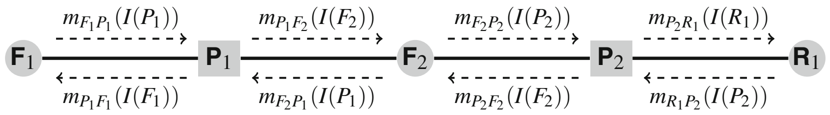


Fig. 4. Message passing of loop belief propagation on the pMN shown in Fig. 3

The loopy belief propagation works as follows. Each node sends messages to its adjacent nodes, as shown in Fig. 4, according to

$$m_{ij}(x_j) = \sum_{x_i \in \{low,high\}} \pi_i(x_i)\Psi_{ij}(x_j|x_i) \prod_{k \in N(i)\backslash j} m_{ki}(x_i), \tag{13}$$

where $m_{ij}(x_j)$, $i \in N(j)$ indicates the message from adjacent nodes of node j, $\pi_i(x_i)$ is the prior of node i. $\Psi_{ij}(x_i|x_j)$ is the conditional probability of integrity level of x_i given that of x_j, which is derived as follows.

$$\Psi_{ij}(x_i|x_j) \propto P(x_i,x_j) = \hat{P}(J_I(x_i,x_j)) = \sum_{E_I(x_i,x_j)} P(J(x_i,x_j)|E_I(x_i,x_j))P(E_I(x_i,x_j)|Acc). \tag{14}$$

Substituting Eqs. (1)–(3), (9)–(11) into Eq. (14), we can easily derive the conditional probability $\Psi_{ij}(x_i|x_j)$. The message $m_{ij}(x_j)$ reveals that how the node i thinks about the level of node j. In each iteration, the message of all nodes will be updated. The order of message updating is not important. The iteration stops when the message of nodes converge, i.e., there is no significant changes of messages between iterations, or when a sufficient number of iterations is reached. Then, we compute the marginal distribution of integrity for each node, a.k.a. the belief of node, as follows.

$$b_i(x_i) = C\pi_i(x_i) \prod_{k \in N(i)} m_{ki}(x_i), \tag{15}$$

where C is a normalization constant to ensure that the integrity probabilities add up to 1, i.e., $\sum_{x_i \in \{L,H\}} b_i(x_i) = 1$. Here, $b_i(L)$ and $b_i(H)$ indicate probabilities that i has *low* and *high* integrity level respectively.

3.4 Malware Detection

There exist violations in commercial modern operating systems under NRD and NWU security policies [21]. In order to accommodate violations, we employ a statistical learning technique to extract more adaptive security policies for malware detection. Before we describe this technique, we first show how to recover the integrity level of subjects and objects under the probabilistic notation.

Integrity Level. The integrity levels of subjects and objects are recovered by taking into account the probability that the subject/object has *high* integrity, i.e., $P(I(i) = H)$, which is the belief of a subject/object $b_i(H)$. We sort subjects and objects by their beliefs of $b_i(H)$ in decreasing order, and assign the integrity level to all subjects and objects from the highest to the lowest. We treat the subjects and objects with same beliefs as having the same integrity level. The ranking positions under the sort are treated as the integrity levels of subjects and objects.

Malware Detection. For program i, we create a feature vector $\mathbf{X}_i$ for it. The feature vector is similar to [20], but with column normalization. More specifically, $\mathbf{X}_i$ is

$$\mathbf{X}_i = \left[\mathbf{x}_i^{(\text{file,read})}, \mathbf{x}_i^{(\text{file,write})}, \mathbf{x}_i^{(\text{reg,read})}, \mathbf{x}_i^{(\text{reg,write})}\right], \tag{16}$$

$$\mathbf{x}_i^{(k,l)} = \left[x_{i1}^{(k,l)}, ..., x_{ir}^{(k,l)}, ..., x_{iL^{(k)}}^{(k,l)}\right]. \tag{17}$$

Here $x_{ir}^{(k,l)}$ is the fraction of objects of type $k \in \{\text{file}, \text{registry}\}$ accessed under operation $l \in \{\text{read}, \text{write}\}$ at integrity level r. $\mathbf{x}_i^{(k,l)}$ is the vector of these fractions at all integrity levels. $L^{(k)}$ is the total number of integrity levels of objects of type k.

Once we create feature vectors for both benign and malicious processes, we train a statistical classifier to build an access behavior model for malware detection.

3.5 Time Complexity

The main time complexity of malware detection model consists of two parts: (1) Integrity determination. (2) Malware classification.

To analyze the time complexity of the first part, we assume the number of edges is E in our pairwise Markov network, which corresponds to the number of subject-object pairs. Our model employ loopy belief propagation, which is a iterative algorithm running in $O(4E(s+1))$, where s is the number of iterations. In each iteration, we need to calculate messages on both directions of each edge, and each direction contains two types of messages, i.e., L and H. That is the reason of the constant before the number of edges. We observe $s \ll E$[1]. Hence,

[1] In fact, we find s is about 7 in our experiments.

we can say, our probabilistic integrity model run in the linear time to the number of edges in our pairwise Markov network, i.e., $O(E)$.

The time complexity of malware classification depends on the statistical classifier we employ, we analyze it in our experimental results.

4 Experimental Evaluation

4.1 Evaluation Methodology

To evaluate our model, we design a set of experiments to empirically answer three questions: (1) Do the determined integrity levels support malware detection from a perspective of security policies? (2) What is the performance of our malware detection model? (3) What is the running time of our probabilistic integrity model in experiments?

Experimental Settings. *Benign programs*: We employ Process Monitor [22] to collect the access behaviors of programs under eight different users' normal usages without interfering with their daily usages, which run on systems running Microsoft Windows XP SP3. Among eight users, two of them are male undergraduates who were working on their final year projects, and six others are graduates consisting of one female student and five male students, whose behaviors include writing, programing, web surfing, etc. The data collection takes place over periods of 7 to 16 days, and we finally obtain access behaviors of 27,840 executions from 534 benign programs.

Malicious programs: We download a collection of 270 K malware samples from VxHeaven, which is a website providing information about viruses [30]. We randomly select 9 K samples, run them in our sandbox which is with Windows XP SP3 on VMWare without network connection, and monitor their behaviors with Process Monitor. After running each sample for five minutes, we revert the virtual machine to a clean snapshot so that different samples do not interfere with each other. Since not all samples exhibit file or registry access activities, we finally obtain access behaviors of 7,257 malware samples. The families of final malware samples are listed in Table 1.

Table 1. Number of malware samples in each family

Family	Samples	Family	Samples	Family	Samples	Family	Samples
Backdoor	25	Trojan-Banker	75	Trojan-Clicker	37	Trojan-PSW	216
Trojan-Spy	768	Trojan-Dropper	128	Trojan-GameThief	278	Trojan-IM	3
Trojan-Mailfinder	5	Trojan-Ransom	2	Trojan.Win32	575	Virus.BAT	1
Virus.JS	3	Virus.MSIL	18	Virus.MSWord	3	Virus.Multi	24
Virus.NSIS	1	Virus.Win32	2527	Virus.WinHLP	5	Worm.BAT	12
Worm.MSIL	4	Worm.Win32	2547				

Training and testing sets: Because the executions of benign programs are collected from eight users, we set up eight experiments. In each experiment, we select the executions of benign programs from one user as a benign testing set, and those from seven other users as benign training set. We infer the probabilistic integrity level of objects from the benign training set. Meanwhile, we randomly select 80 % malware samples as a malicious training set and the remaining ones as a malicious testing set. We repeat the experiment 20 times. The results of 20 repetitions are averaged and illustrated in the following subsections.

Hyperparameters: To avoid priors dominating the data in our generative model as shown in Eqs. (4) and (6), we choose Jeffreys priors, i.e., α_1, α_2, α_3, β_1, β_2, and β_3 are equal to 0.5, which are non-informative priors and invariant under transformation [9].

New objects: It is very common to come across new objects which do not appear in the training set. For example, objects are newly created by processes in the testing set. We employ a heuristic method to assign probabilistic integrity level to new objects, which is similar to the approach in [20], but from a probabilistic perspective. More specifically, we assign probabilistic integrity level to new objects according to the directory it is stored in. The probability that the new object has high integrity level equals to the probability that its parent directory has high integrity level, which equals to the highest probability that the child objects of the parent directory have high integrity level.

Statistical classifier: We employ random forests, implemented in Scikit-learn [24], as our classifier. Random forests is an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time. It not only exhibits the same advantage of decision trees, but also overcomes the main disadvantage of decision trees, namely overfitting [5]. Moreover, we observe that the results are comparable in performance to other classifiers, e.g., k-nearest neighbors, logistic regression, and support vector machine. We do not demonstrate them in this paper considering page limits.

Baseline. We compare our method to two baseline models for determining integrity levels. The first baseline model ($B1$) strictly obeys NRD and NWU, which forms a lattice constructed from access behaviors of benign programs. The lattice consists of partial orders of integrity levels between subjects and objects determined by observed access events under NRD and NWU. The layers in the hierarchical structure of the lattice indicate integrity levels. Figure 5 illustrates the lattice constructed from our data set of benign programs. We observe four layers in this hierarchical structure, which indicates four possible integrity levels of subjects and objects. The lattice is shown as integrity levels increasing from bottom to top. We employ $B1$ because it is a mandatory integrity protection, which relies solely on NRD and NWU security policies.

The second baseline model ($B2$) is the importance based malware detection model introduced in [20]. It assigned importance values to all subjects and objects by examining their structures in a dependency network, and employed statistical classifiers to detect malware based on the assigned importance values

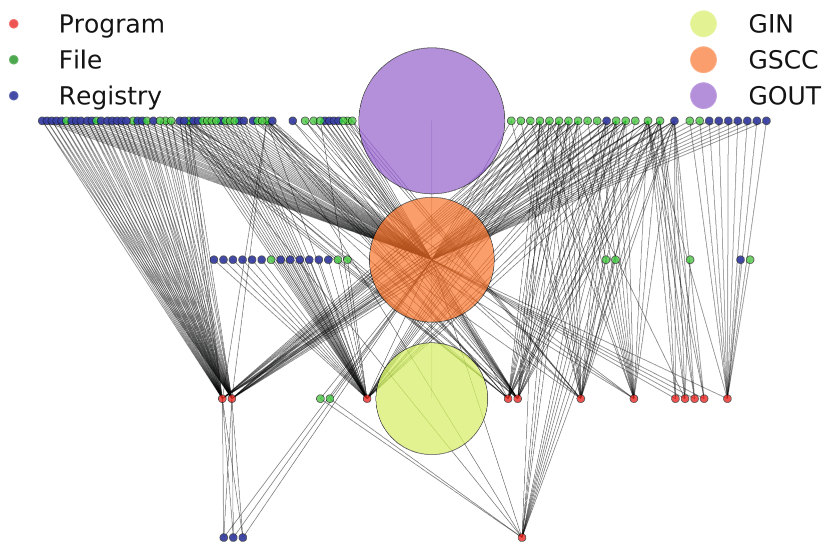


Fig. 5. The lattice constructed from access events of benign programs in our data set. GSCC stands for giant strongly connected component, GIN represents in-neighbors of the GSCC but without any edge from the GSCC, and GOUT represents out-neighbors of the GSCC but without any edge to the GSCC. Each small node indicates either a subject (program) or an object (file or registry).

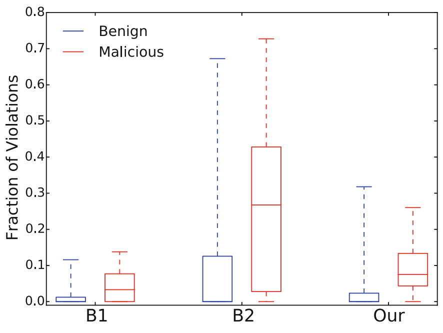
of objects. The reason for choosing B2 is that this paper has similar goal with that, although there is a gap between importance values and integrity levels.

4.2 Integrity Levels and Security Policies

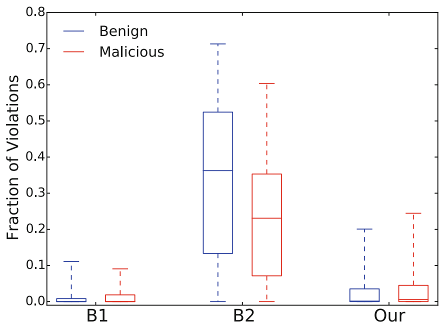
To explore the appropriateness of the derived integrity levels, we examine the differences exhibited between benign and malicious programs from the perspective of their compliance to security policies NRD and NWU. That is, we investigate the violations of security policies for benign and malicious processes based on the derived integrity levels. We define a violation of NRD or NWU when a process reads an object with low integrity level, and writes an object with high integrity level.

Let p be a process, O_r be the set of reading objects of p, O_w be the set of writing objects of p, $o_r \in O_r$, $o_w \in O_w$. Strictly, according to the NRD and NWU policies, the integrity levels of p, o_r and o_w should satisfy $I(p) \leq I(o_r)$ and $I(p) \geq I(o_w)$, i.e., $I(p) - I(o_r) \leq 0$ and $I(o_w) - I(p) \leq 0$. Because of difficulty in determining $I(p)$, we employ a proxy $I(o_w) - I(o_r) \leq 0$, by summing up the two criteria, to examine violations. A violation happens to the process p when $I(o_w) - I(o_r) > 0$. We refer to a *violation* as a pair of reading and writing objects where the integrity level of the reading object is lower than that of the writing object. If the determined integrity levels of objects are correct, few violations would appear in benign processes, while many violations would appear in malicious processes.

We examine violations in processes by two simple indicators based on our proxy for violation: (1) *Fraction of violation.* In each execution, we count the fraction of violations among all pairs of reading objects and writing objects, e.g., $\frac{\sum_{o_r,o_w} \mathbb{1}(I(o_w)-I(o_r)>0)}{|O_r||O_w|}$, where $\mathbb{1}()$ is an indicator function, $|O_r|$ and $|O_w|$ are the sizes of set O_r and O_w. (2) *Largest violation.* This refers to the difference between the lowest integrity level of all reading objects and the highest integrity level of all writing objects in one execution, e.g., $\max_{o_r,o_w} I(o_w) - I(o_r)$. These two simple indicators demonstrate why the integrity levels of objects are useful to detect malware from a perspective of security policies. Figure 6 exhibits fraction of violations, while Fig. 7 illustrates largest violations, for benign and malicious processes w.r.t. integrity levels of access objects determined by baselines and our model, in terms of box plots. These results are obtained from all testing sets of eight experiments which are presented in the above subsection. A box plot splits these results into quartiles. The interquartile range box represents the middle 50 % of the results. The whiskers, extending from either side of the box, represent the ranges for the bottom 5 % and the top 5 % of the results.



(a) Fraction of violation on file objects

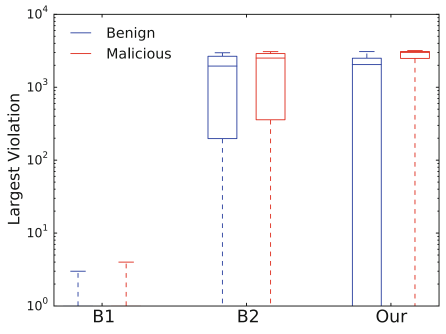


(b) Fraction of violation on registry objects

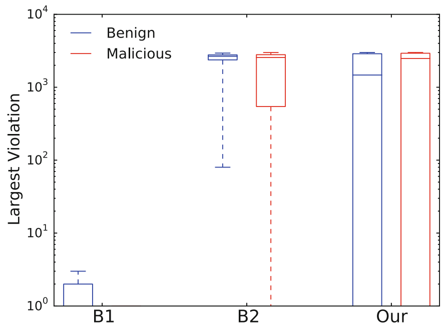
Fig. 6. Fraction of violation under integrity levels from baseline and our models

In Fig. 6(a), as we expected, there are fewer violations in benign processes than in malicious processes, w.r.t. integrity levels of accessed file objects, under all three models. Using the Kolmogorov–Smirnov (KS) test, we find significant differences ($p \ll 10^{-4}$) between benign and malicious processes under all three models in Fig. 6(a). The KS test is a nonparametric test to evaluate whether two samples come from the same population. These results indicate each model can be used to determine integrity levels. We also find that our model is more able to discriminate between benign and malicious processes than the two baseline models, with respect to integrity levels of accessed file objects. This indicates that our model does a better job at determining integrity levels for file objects.

With respect to the integrity levels of accessed registry objects, we do not observe obvious difference between benign and malicious processes in Fig. 6(b), although significant differences ($p \ll 10^{-4}$) are found under KS test. There are two possible reasons: (1) All models fail to determine the integrity levels of registry objects. (2) Benign processes do not obey NRD and NWU policies when they access registry objects.



(a) Largest violation on file objects



(b) Largest violation on registry objects

Fig. 7. Largest violation under integrity levels from baseline and our models

Moreover, we explore the difference between benign and malicious processes according to the largest violation. Figure 7(a) illustrates the largest violations of benign and malicious processes on file objects, while Fig. 7(b) illustrates those on registry objects. Since the total number of integrity levels in B1 is much less than those in other two models, we show the largest violation in logarithmic scale. Similar results are observed in Fig. 7(a) and (b) compared to Fig. 6(a) and (b). Meanwhile, we find significant differences ($p \ll 10^{-4}$) under the KS test in all cases. As shown in Fig. 7(a), our model achieves the greatest discrimination between benign and malicious processes according to integrity levels of accessed file objects. It implies the ability of our model for malware detection even with camouflages. However, we find similar failures of all three models in distinguishing malicious programs from benign programs, which due to similar possible reasons as we aforementioned.

We observe obvious differences between benign and malicious processes by examining either of these two indicators. However, the fraction of violation may suffer from *mimicry* or *camouflaged attacks*, where malicious processes run under the cover of some benign processes [10,15]. For example, if malware deliberately read many file objects with the highest integrity levels, then the numerator in the fraction of violations will be overwhelmed by the denominator in the fraction, which leads to as small fraction of violations as benign processes. Compared with the fraction of violation, the indicator of largest violation is much more robust. However, one potential failure of the largest violation indicator is false positive. There are benign processes which modify objects with high integrity level while read objects with low integrity level. Usually, these processes are some special processes, such as system services. We can include them in a whitelist, and reduce the false positives of the largest violation indicator.

4.3 Detection Results

Although simple indicators, e.g., fraction of violations, largest violation, etc., provide us ways of understanding why a model works, they usually fail to achieve promising performance on malware detection. As presented in Sect. 3.4, we employ random forests to extract adaptive security policies under determined integrity levels and build a model for malware detection. With three models for determining integrity levels, we evaluate their performance on malware detection with the ROC curve and the area under ROC curve (AUC). We train models with the benign and malicious training sets, and evaluate them on the benign and malicious testing sets, as stated in Sect. 4.1.

Table 2 exhibits average true positive rates (TPRs) of three models at specific false positive rates (FPRs) among all experiments. We choose these four FPRs, because they are four representative FPRs to evaluate a method of malware detection in practice. Meanwhile, we emphasis the most outperformed results of our models compared with baseline models, i.e., 99.88 % TPF at 0.1 % FPR, on average. In most cases, our model achieves better performance than two baseline models.

Table 2. Performance under different models of determining integrity level

Model	FPR	True Positive Rate (TPR)								Average
		U1	U2	U3	U4	U5	U6	U7	U8	TPR
B1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	0.1%	92.09%	83.59%	70.30%	76.08%	53.35%	79.79%	64.10%	43.10%	70.30%
	0.5%	96.75%	91.75%	92.55%	88.50%	86.18%	90.31%	90.98%	76.89%	89.24%
	1.0%	98.09%	96.64%	95.18%	92.52%	91.85%	91.54%	94.19%	90.08%	93.76%
B2	0%	99.52%	82.34%	97.58%	98.22%	88.16%	99.40%	0%	99.73%	83.11%
	0.1%	100%	90.02%	99.27%	99.85%	92.92%	99.53%	70.21%	99.73%	93.94%
	0.5%	100%	99.60%	99.86%	100%	99.57%	99.66%	95.32%	99.73%	99.21%
	1.0%	100%	99.93%	100%	100%	99.86%	99.66%	99.16%	99.86%	99.80%
Our	0%	99.45%	99.49%	99.51%	97.15%	98.30%	99.71%	4.48%	98.31%	87.05%
	0.1%	99.98%	99.98%	99.95%	99.89%	99.87%	99.94%	99.61%	99.85%	99.88%
	0.5%	99.99%	100%	99.98%	99.98%	99.97%	99.98%	99.89%	99.93%	99.97%
	1.0%	100%	100%	99.98%	99.99%	99.99%	99.99%	99.94%	99.97%	99.98%

To further compare the performance of the three models, we conduct a Wilcoxon rank-sum test to evaluate whether one model significantly outperforms the other in terms of AUC. The Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used when comparing two related samples to assess whether their population mean ranks differ. Table 3 illustrates the test statistic and its significance between each pair of models. A negative value of the test statistic indicates that the first model performs worse than the second model shown at the beginning of the row.

Table 3. Results of Wilcoxon rank-sum tests on AUCs of different models

Models	U1	U2	U3	U4	U5	U6	U7	U8	All
B1 v.s. B2	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-15.47**
B1 v.s. Our	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-5.41**	-15.47**
B2 v.s. Our	-2.89*	-5.41**	-0.14 (0.89)	-0.81 (0.41)	-2.84*	-0.77 (0.44)	-3.68**	-4.25**	-5.44**

* Indicates significance under 0.01.
** Indicates significance under 0.0001.
p-value is shown in parenthesis if it is not significant under these two levels.

In Table 3, from column U1 to U8, we perform hypothesis testing on results of 20 runnings within each experiment. The last column presents the result of hypothesis testing on results of runnings of all experiments. We observe significant improvements of our model compared with baseline models.

Furthermore, Fig. 8 illustrates average ROCs of eight experiments under our probabilistic integrity level, which provides better understandings of the performance.

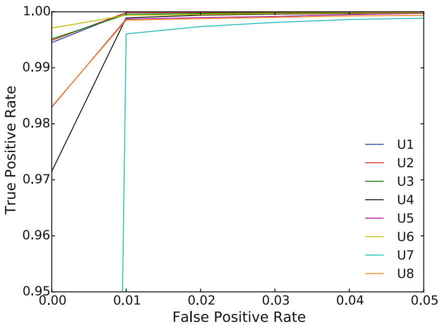


Fig. 8. Average ROC of eight experiments under our model

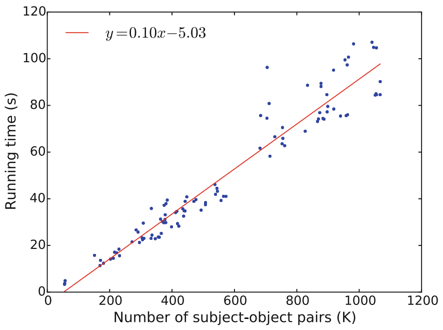


Fig. 9. Scatter plot of running time in seconds against problem size in thousands

4.4 Running Time

The time complexity of the employed classifier, random forests, has been well studied, which is, $mn \log n$, where m is the number of trees in random forests, we find $m = 10$ is optimal in our experiments, n is the number of processes, a.k.a. data points [5]. Hence, we do not present its running time.

We explore the running time of our probabilistic integrity model, since the running time of loopy belief propagation in practices varies in different problems. We vary the problem size, which is the number of subject-object pairs, by randomly selecting different portions of subjects and their involving subject-object pairs. Figure 9 illustrates a scatter plot of running time in seconds against problem size in thousands, and its fitting with linear regression. We observe strongly linear relationship, supported by significant coefficients and $R^2 = 0.93$ in the linear regression, between the running time and the problem size. This result verifies the linear time complexity of our probabilistic integrity model, and demonstrates the feasibility of a runtime malware detection.

5 Conclusion and Future Work

In spite of considerate effort by security researchers and engineers, it has been demonstrated that attackers move faster than defenders. This paper presents a probabilistic model on access behaviors of programs, and integrity levels of programs, files and registries. We employ probabilistic inferences to determine integrity levels of these system subjects and objects. Combining with a statistical classifier, we build a integrity based access behavior model for malware detection. The encouraging experimental results indicate the feasibility and usefulness of our model. The linear time complexity of our probabilistic integrity model is both proofed by our theoretical analysis, and verified by our experimental results.

Our model can be extended to subject and objects in other granularities, which are constrained by similar security policies. Meanwhile, our model can be adapted to determine levels of other security attributes, e.g., confidentiality, according to corresponding security policies, e.g., Bell-LaPadula model.

We believe our probabilistic integrity model will be enhanced, when acquiring knowledge from both benign and malicious programs. Thus, building a model to combine access behaviors of both benign and malicious programs will be our future work.

Acknowledgments. We would like to thank our shepherd, Manos Antonakakis, and the anonymous reviewers for their insightful comments that greatly helped improve the presentation of this paper. This work is supported by NFSC (61175039, 61221063, 61403301), 863 High Tech Development Plan (2012AA011003), Research Fund for Doctoral Program of Higher Education of China (20090201120032), International Research Collaboration Project of Shaanxi Province (2013KW11) and Fundamental Research Funds for Central Universities (2012jdhz08). Any opinions, findings, and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

Appendix- Derivation of Eq. (8)

$P(E_I|Acc) \propto \sum_T P(Acc|T)P(T|E_I) \sum_D P(E_I|D)P(D)$, where

$$\sum_D P(E_I|D)P(D) = \begin{cases} \sum_D d_1 P(D) = \mathbb{E}_D(d_1) = \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3}, & \text{if } I(s) < I(o), \\ \sum_D d_2 P(D) = \mathbb{E}_D(d_2) = \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3}, & \text{if } I(s) = I(o), \\ \sum_D d_3 P(D) = \mathbb{E}_D(d_3) = \frac{\alpha_3}{\alpha_1+\alpha_2+\alpha_3}, & \text{if } I(s) > I(o). \end{cases} \quad (18)$$

And then,

(1.) If $I(s) < I(o)$:

$$\begin{aligned} P(<|Acc) &\propto \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3}\Delta\int_T t_1^{N_r}t_2^{N_w}t_3^{N_{r\&w}}\frac{t_1^{\beta_1}t_2^{\beta_2-1}t_3^{\beta_3-1}}{B(1+\beta_1,\beta_2,\beta_3)}\,\mathrm{d}T,\\ &= \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3}\Delta\frac{B(N_r+\beta_1+1,N_w+\beta_2,N_{r\&w}\beta_3)}{B(1+\beta_1,\beta_2,\beta_3)},\\ &= \frac{\alpha_1}{\alpha_1+\alpha_2+\alpha_3}\Delta\frac{\beta_1+\beta_2+\beta_3}{\beta_1}\frac{N_r+\beta_1}{N+\beta_1+\beta_2+\beta_3}\Omega, \end{aligned} \tag{19}$$

where $\Delta = \frac{\Gamma(N+1)}{\Gamma(N_r+1)\Gamma(N_w+1)\Gamma(N_{r\&w}+1)}$, $\Omega = \frac{B(N_r+\beta_1,N_w+\beta_2,N_{r\&w}+\beta_3)}{B(\beta_1,\beta_2,\beta_3)}$, and $B(\beta_1,\beta_2,\beta_3) = \frac{\Gamma(\beta_1)\Gamma(\beta_2)\Gamma(\beta_3)}{\Gamma(\beta_1+\beta_2+\beta_3)}$.

(2.) If $I(s) = I(o)$:

$$P(=|Acc) \propto \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3}\Delta\int_T t_1^{N_r}t_2^{N_w}t_3^{N_{r\&w}}\frac{t_1^{\beta_1-1}t_2^{\beta_2-1}t_3^{\beta_3-1}}{B(\beta_1,\beta_2,\beta_3)}\,\mathrm{d}T = \frac{\alpha_2}{\alpha_1+\alpha_2+\alpha_3}\Delta\Omega. \tag{20}$$

(3.) If $I(s) > I(o)$:

$$\begin{aligned} P(>|Acc) &\propto \frac{\alpha_3}{\alpha_1+\alpha_2+\alpha_3}\Delta\int_T t_1^{N_r}t_2^{N_w}t_3^{N_{r\&w}}\frac{t_1^{\beta_1-1}t_2^{\beta_2}t_3^{\beta_3-1}}{B(\beta_1,\beta_2+1,\beta_3)}\,\mathrm{d}T,\\ &= \frac{\alpha_3}{\alpha_1+\alpha_2+\alpha_3}\Delta\frac{\beta_1+\beta_2+\beta_3}{\beta_2}\frac{N_w+\beta_2}{N+\beta_1+\beta_2+\beta_3}\Omega. \end{aligned} \tag{21}$$

Summing up Eqs. (19)–(21), we derive the posterior distribution of E_I given Acc, i.e., $P(E_I|Acc)$, as shown in Eqs. (9)–(11).

References

1. Anderson, R.: Security Engineering: A Guide to Building Dependable Distributed Systems. John Wiley & Sons (2008)
2. Apap, F., Honig, A., Hershkop, S., Eskin, E., Stolfo, S.J.: Detecting malicious software by monitoring anomalous windows registry accesses. In: Wespi, A., Vigna, G., Deri, L. (eds.) RAID 2002. LNCS, vol. 2516, p. 36. Springer, Heidelberg (2002)
3. Bellovin, S.M.: Security and usability: windows vista, July 2007. https://www.cs.columbia.edu/smb/blog/2007-07/2007-07-13.html
4. Biba, K.J.: Integrity considerations for secure computer systems. ESD-TR 76–372, MITRE Corp. (1977)
5. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
6. Canali, D., Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., Kirda, E.: A quantitative study of accuracy in system call-based malware detection. In: Proceedings of the 2012 International Symposium on Software Testing and Analysis, pp. 122–132. ACM (2012)
7. Fraser, T.: Lomac: low water-mark integrity protection for cots environments. In: IEEE Symposium on Security and Privacy (S&P), pp. 230–245 (2000)

8. Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., Yan, X.: Synthesizing near-optimal malware specifications from suspicious behaviors. In: IEEE Symposium on Security and Privacy (S&P), pp. 45–60 (2010)
9. Gelman, A., Carlin, J.B., Stern, H.S., Rubin, D.B.: Bayesian data analysis, vol. 2. Taylor & Francis (2014)
10. Gu, Z., Pei, K., Wang, Q., Si, L., Zhang, X., Xu, D.: LEAPS: detecting camouflaged attacks with statistical learning guided by program analysis. In: 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE (2015)
11. How the integrity mechanism is implemented in Windows Vista (2014). http://msdn.microsoft.com/en-us/library/bb625962.aspx,
12. Hsu, F., Chen, H., Ristenpart, T., Li, J., Su, Z.: Back to the future: a framework for automatic malware removal and system repair. In: 22nd Annual Computer Security Applications Conference, ACSAC 2006, pp. 257–268. IEEE (2006)
13. King, S.T., Chen, P.M.: Backtracking intrusions. ACM Trans. Comput. Syst. **23**, 51–76 (2005)
14. Koller, D., Friedman, N.: Probabilistic graphical models: principles and techniques. MIT press (2009)
15. Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Automating mimicry attacks using static binary analysis. In: Proceedings of the 14th conference on USENIX Security Symposium, vol. 14, pp. 11–11. USENIX Association (2005)
16. Kruegel, C., Mutz, D., Valeur, F., Vigna, G.: On the detection of anomalous system call arguments. In: Snekkenes, E., Gollmann, D. (eds.) ESORICS 2003. LNCS, vol. 2808, pp. 326–343. Springer, Heidelberg (2003)
17. Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., Kirda, E.: Accessminer: using system-centric models for malware protection. In: Proceedings of the 17th ACM conference on Computer and Communications Security (CCS), pp. 399–412. ACM (2010)
18. Manadhata, P.K., Yadav, S., Rao, P., Horne, W.: Detecting malicious domains via graph inference. In: Kutyłowski, M., Vaidya, J. (eds.) ICAIS 2014, Part I. LNCS, vol. 8712, pp. 1–18. Springer, Heidelberg (2014)
19. Mandatory Integrity Control (2014). http://msdn.microsoft.com/en-us/library/windows/desktop/bb648648
20. Mao, W., Cai, Z., Guan, X., Towsley, D.: Centrality metrics of importance in access behaviors and malware detections. In: Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC 2014). ACM (2014)
21. Mao, Z., Li, N., Chen, H., Jiang, X.: Combining discretionary policy with mandatory information flow in operating systems. ACM Trans. Inf. Syst. Secur. (TISSEC) **14**(3), 24 (2011)
22. Mark Russinovich, B.C.: Process monitor (2014). http://technet.microsoft.com/en-us/sysinternals/bb896645
23. Muthukumaran, D., Rueda, S., Talele, N., Vijayakumar, H., Teutsch, J., Jaeger, T., Edwards, N.: Transforming commodity security policies to enforce Clark-Wilson integrity. In: Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012). ACM (2012)
24. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
25. Sun, W., Sekar, R., Liang, Z., Venkatakrishnan, V.N.: Expanding malware defense by securing software installations. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 164–185. Springer, Heidelberg (2008)

26. Sun, W., Sekar, R., Poothia, G., Karandikar, T.: Practical proactive integrity preservation: a basis for malware defense. In: IEEE Symposium on Security and Privacy (S&P), pp. 248–262 (2008)
27. Symantec. Internet Security Threat Report, April 2015. https://www4.symantec.com/mktginfo/whitepaper/ISTR/21347932_GA-internet-security-threat-report-volume-20-2015-social_v2.pdf
28. Sze, W.-K., Sekar, R.: A portable user-level approach for system-wide integrity protection. In: Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC 2013), pp. 219–228. ACM (2013)
29. Tamersoy, A., Roundy, K., Chau, D.H.: Guilt by association: large scale malware detection by mining file-relation graphs. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge Discovery and Data Mining, pp. 1524–1533. ACM (2014)
30. VXHeaven (2010). http://vx.netlux.org/