Hindawi
Journal of Electrical and Computer Engineering
Volume 2018, Article ID 9250297, 13 pages
https://doi.org/10.1155/2018/9250297

Research Article

Behavior Intention Derivation of Android Malware Using Ontology Inference

Jian Jiao[1,2] **Qiyuan Liu**[1,2] **Xin Chen**[2] **and Hongsheng Cao**[1,2]

[1]*Beijing Key Laboratory of Internet Culture and Digital Dissemination Research, Beijing Information Science and Technology University, Beijing, China*
[2]*School of Computer Science, Beijing Information Science and Technology University, Beijing, China*

Correspondence should be addressed to Jian Jiao; jiaojian@bistu.edu.cn

Received 2 November 2017; Revised 26 January 2018; Accepted 20 February 2018; Published 1 April 2018

Academic Editor: Ahmad K. Malik

Previous researches on Android malware mainly focus on malware detection, and malware's evolution makes the process face certain hysteresis. The information presented by these detected results (malice judgment, family classification, and behavior characterization) is limited for analysts. Therefore, a method is needed to restore the intention of malware, which reflects the relation between multiple behaviors of complex malware and its ultimate purpose. This paper proposes a novel description and derivation model of Android malware intention based on the theory of intention and malware reverse engineering. This approach creates ontology for malware intention to model the semantic relation between behaviors and its objects and automates the process of intention derivation by using SWRL rules transformed from intention model and Jess inference engine. Experiments on 75 typical samples show that the inference system can perform derivation of malware intention effectively, and 89.3% of the inference results are consistent with artificial analysis, which proves the feasibility and effectiveness of our theory and inference system.

1. Introduction

Android malware forms a gray interest industrial chain for the purpose of tariff consumption, malicious chargeback, malicious promotion, and privacy trafficking (privacy theft). Security companies face a large number of suspicious samples to analyze every day. The workload of manual analysis is huge, and the extraction efficiency of malicious features is low, which worsens the security situation of Android [1]. Therefore, the analysis of malware samples has become a top priority of Android security. Traditional malware analyses rely on the analysts' experience, and analysts can cope with limited number of new malicious samples, while being unable to handle it when the scale is too large. In addition, though detection tools based on malware characteristics and behavior patterns can detect some malicious behavior of malware, there are still some problems:

(i) The detected results only give a preliminary judgment of malware's malice. For example, this application may cause your privacy to be stolen or your cost loss. But they did not explain what kind of privacy was stolen and which kind of fee deduction happened.

(ii) Detection of malicious behavior may not reflect the ultimate intention of malware, such as remote control.

(iii) The detected information of behavior is limited and the granularity of the hint is too thick to carry out the whole malicious software intention analysis, and there are also some uncertain factors such as false positive rate.

Therefore, the traditional methods of malware detection combined with manual analysis are facing severe challenges. It has become the bottleneck of the development of Android security to some extent. We need to transform the traditional thinking pattern of research, focus on how to present malicious software intention to ordinary users, and allow users to obtain high readable software analysis results.

Behavior intention is defined as a *sequence of software behaviors* directed at a particular *purpose*. The definition contains two key components: sequence of software behaviors and purpose. We think that sensitive behaviors can be extracted from a program using sensitive APIs invocation

(security relevant API invocation, usually controlled by sensitive permission) as a clue. But what is the purpose? How to represent the purpose of malware to users? We use one of our examples to illustrate the problem. This malware involves sensitive API calls, such as getMessageBody() and getDeviceId(), setEntity(), connect(), and execute(). It first accesses user's short message and device ID number; after encrypting them, it connects to a remote server and transmits the information to the server. We use access, encrypt, connect, and transmit to represent these behaviors, but the semantic of access (behavior) depends on its object (SMS, DeviceId); thus we use these objects to represent behaviors. After these APIs execution, the state of the objects will change; we use the final states of these objects to describe the purpose of intention. Now the key components in the definition of intention have been illustrated. We use this idea as motivation example to put forward our theory.

The aims of this research are as follows: firstly, extracting security sensitive semantic layer behaviors depending only on the semantic information (sensitive APIs) of the program; secondly, proposing a derivation and description method of security centric behavior chain that does not care about unrelated software behavior; thirdly, the final intention results based on the combination of natural language and formal representation. The advantage of formalization is that automatic behavior reasoning can be implemented and natural language description of behaviors can help users understand the semantics of program. The reasoning results can be represented by graphical method easily, so that the end-users can get concise and intuitive software intention information.

We propose a new description and derivation model of malware intention using the existing reverse analysis results [2–4] of Android malware and the theory of intention [5, 6]. On this basis, we complete the deduction of behavioral intention at semantic level with the help of ontology reasoning technology [7] and realize the reverse reduction from low-level Android malicious code to high-level intention finally. The main work of this paper is divided into two aspects: Firstly, the definition and formalized model of malware intention are given and the rationality of this model is proved. Secondly, we use ontology to model the semantic relationship between behaviors and objects and automate the process of intention derivation using SWRL [8] rules and Jess engine [9]. SWRL is a semantic web rule language combining OWL and RuleML and also a language that presents rules in a semantic way. Jess engine is the rule engine for Java platform, which can be used to complete many complex logical reasoning tasks.

The reasons for introducing ontology are as follows: (1) the conceptual system based on ontology is computable, and the automatic deduction of intentions can be achieved by the computing between concepts; (2) the ontology model of the elements and their relationships in malware behavior intention domain can standardize domain knowledge and make it shareable [10]; (3) the extensibility of ontology enables us to extend the conceptual model at any time according to the emergence of new features of malware.

The rest of the paper is organized as follows. Section 2 introduces related works about the research status of malware detection, reverse analysis, and the theory of intention. Section 3 elaborates the modeling process of the intention. Section 4 defines ontology model and SWRL rule and describes the establishment process of Fact Base. Section 5 shows the experimental results of our representative sample of malware. Finally, the paper concludes with Section 6.

2. Related Works

The essence of malware intention reduction is behavior analysis and software intention is divorced from the specific API function call. It is a behavior abstraction of semantic layer and can describe the behavior semantics intuitively. From the perspective of reverse engineering, ApkRiskAnalyzer [11] uses commercial disassembly tool IDA Pro [12] to collect the disassembly information. It constructs taint analysis engine and constant analysis engine to detect privacy disclosure behavior and constant use malicious behavior based on collected information. Jian and Qing [13] use the program slice of sensitive API to obtain the calling sequence of malicious code and then return the use case diagram of the whole malicious code. They restore the functional semantics of malicious behavior to some extent and help users locate malicious code quickly, but they do not consider the specific relationship between behaviors and cannot reflect the real intention of malware.

Android malware research mainly focuses on the determination of malicious behavior and the extraction of malicious features. Although the semantic relations between APIs are considered in [14–16] and certain results have been achieved, they come to the direction of malware detection eventually and cannot provide complete analysis and presentation of malware intention. AppContext [14] judges whether the behavior is benign or malicious according to the context of the sensitive API. It is believed that the trigger of security sensitive behavior can be judged by context, such as trigger events that lead to sensitive behavior or external environmental conditions. DroidADDMiner [15] gives the construction method of feature vectors required for machine learning through the extraction of data dependence paths between sensitive APIs and combines constant usage and context information, which can detect, classify, and characterize Android malware eventually. DroidSIFT [16] constructs weighted contextual API dependency graph based on semantics and then gives the corresponding weight according to the risk of sensitive API. After constructing the data set of behavior graph, similarity value of graph is used as the element of machine learning feature vector and the feature construction is completed finally.

The research of intention recognition is widely used in network security defense, artificial intelligence, natural language understanding, and so on. Intrusion intention recognition [17] is an analysis of a large amount of underlying alarm information to explain and determine what attackers want to achieve. It is essentially a process of implementing a reasonable interpretation of a large number of attack data. The identification of attacker's intention can

determine the true intention of attackers and predict the subsequent behavior of attackers. It is the premise and foundation of threat analysis and decision response and is an important component of network security situational awareness. Shirley and Evans [6] get the malicious judgment of software behavior according to the matching degree of user's operation intention and software behavior. In the field of artificial intelligence [18], the human intention identified by agents is that of formal representation. These studies provide a theoretical basis for our malware intention modeling.

3. Model of Malware Intention

This chapter focuses on the modeling process of malware intention. First, the formalized definition of software intention is given according to the abstract of the problem and related literature. Then the key component in the definition of intention has been defined and explained. Finally, the inference process of intention model has been proved using mathematical theorem.

3.1. Formalization of Software Intention. In the study of behavioral theory, Bratman Michael [5] believes that intention should be a basic unit of behavior research and suggests that intention is behavior sequence based on future orientation; that is, the intention will influence the behavior of the next step. After the synthesis and analysis of a large number of definitions of intention, the connotation of intention as a sequence of actions directed at a particular purpose is obtained.

To obtain the formalized definition of software intention, program is regarded as combination of data structure and algorithm. Then the problem can be abstracted as follows: Abstracting a function call or program operation of a sensitive API into an action, a sequence of actions that contain specific relations can perform certain functions. They operate on a set of input data, including event messages, user input, privacy data, and constant arguments. The semantics of input data are abstracted into an object description of action. When trigger condition is satisfied a malware starts its process of operations. This set of data (objects) reaches a certain state (data in final) after a series of operations eventually while these final states reflect the purpose of the program's execution.

Definition 1. Software intention is a sequence of software behaviors directed at a particular purpose, formally represented as

$$\begin{aligned}&\overrightarrow{\omega}\,(\text{Intention})\\&\quad=\left(\left\langle\omega_1\left(P_1\right),\omega_2\left(P_2\right),\ldots,\omega_n\left(P_n\right)\right\rangle,\text{Goal}\right).\end{aligned}\tag{1}$$

In formula (1), "$\overrightarrow{\omega}$" refers to directionality, which indicates the behavior sequence $\langle\omega_1(P_1),\omega_2(P_2),\ldots,\omega_n(P_n)\rangle$ points to the purpose Goal; ω represents temporal and spatial attributes of behavior, and temporal attributes describe specific temporal relationships between actions; spatial attributes indicate the impact of behavior on external objects.

3.2. Intention Elements. The elements of software intention include software behavior sequence, behavior objects (behavior facts), and software goals, which are defined below separately. They are the theoretical basis for the derivation of Section 3.3, and the extraction method of behavior facts is described in Section 4.3.

Definition 2. Behavior object is an abstract description of the data or device objects that Android API invocation can operate on. It is a three-tuple:

$$\text{Beh\_object}=\left(\text{obj\_name},\text{attribute},\text{attri\_value}\right)\tag{2}$$

In tuple (2), obj_name represents object name, attribute refers to the object's property, and attri_value represents the property value. Different behaviors act on different objects, and these objects correspond to different entities in the program.

Behavior objects usually presented in the form of a class object (system resource management, network connection objects, and network buffer objects), constants (SP number, advertising link, URL, and file address), local executable files (system command, local library (.so) file), and passed variables (parameter, data path). The abstract semantics of these objects can be determined by certain mapping rules.

Definition 3. Software behavior is an abstract software operation that can operate on the object and change its property value, formally represented as

$$\begin{aligned}\text{Behavior}&=\left(\text{beh\_name},\text{Input},\text{Output}\right)\\\text{Input}&=\left\{\text{Beh\_object}\right\}\\\text{Output}&=\left\{\text{Beh\_object}'\right\}.\end{aligned}\tag{3}$$

In formula (3), beh_name represents the name of a behavior, Input represents the input object descriptions of this behavior, and Output represents the output object descriptions of this behavior. These behaviors can be divided into two types temporarily according to the number of their input in this paper: B1 (single input, single output); B2 (double input, double output).

We have discovered 102 significant behaviors involving 115 sensitive APIs, via sensitive API mining in Genome's Android apps and the summary of related literature [15, 19–21]. We also refer to Android's official API document and a database of sensitive API behavior is established based on the information collected. These APIs are controlled by 30 Android sensitive permissions and have detailed descriptions in official documents.

We generate behavior facts for API patterns based on their internal program logics. There are APIs whose semantics are similar and can be classified as a class of behavior. We further study the extraction mechanism; Table 1 presents the three major logics that we used. (1) Sequential relation APIs constitute a special workflow. For instance, connect() always happens before execute(), since the first provides the

Table 1: The rules of behavior semantic extraction.

Program structure	Location of behavior semantic
(1) Sequential relation	Extracting both of these APIs.
(2) Prepared for the latter	Extracting the latter API.
(3) Multilevel data access	Extracting the former API.

second with necessary inputs. In this case, we study the document of both APIs and extract behaviors from both of them. (2) A former object is retrieved for latter operations. For example, a SmsMessage.createFromPdu(pdus) is always invoked prior to SmsMessage.getMessageBody() because the former fetches the default object SmsMessage that the latter needs. We then only extract behaviors from the latter APIs. (3) Multilevel data is accessed through multiple levels of APIs. For example, when accessing location data, we first call getLastKnownLocation() to return a location object and then call getLongitude() and getLatitude() to get the longitude and latitude from this object. As these higher level APIs are meaningful enough, we hence only extract behaviors from the former APIs.

According to the example analysis of malware samples and the description of related literature, we summarize a variety of malware behaviors and give the behavior set E:

$$\begin{aligned} E ::= \{&\varphi_{\text{right\_gain}}, \varphi_{\text{access}}, \varphi_{\text{store}}, \varphi_{\text{encrypt}}, \varphi_{\text{monitor}}, \varphi_{\text{intercept}}, \\ &\varphi_{\text{connect}}, \varphi_{\text{transmit}}, \varphi_{\text{send}}, \varphi_{\text{dial}}, \varphi_{\text{decode}}, \varphi_{\text{install}}, \varphi_{\text{popup}}, \\ &\varphi_{\text{tamper}}, \varphi_{\text{delete}}, \varphi_{\text{hide}}, \varphi_{\text{remote\_control}}\}. \end{aligned} \tag{4}$$

Taking the monitoring and interception of broadcast message as an example explains how behavior is described. Behavior can be described as a mapping relationship:

(i) Broadcast message monitoring:

$$\begin{aligned} &\varphi_{\text{monitor}} : \text{Broadcast\_info} \longrightarrow \text{Broadcast\_info}' \\ &\text{Broadcast\_info} = (\text{Broadcast\_info}, \text{is\_monitored}, \text{No}) \\ &\text{Broadcast\_info}' \\ &\quad = (\text{Broadcast\_info}, \text{is\_monitored}, \text{Yes}). \end{aligned} \tag{5}$$

(ii) Broadcast message interception:

$$\begin{aligned} &\varphi_{\text{intercept}} : \text{Broadcast\_info}' \longrightarrow \text{Broadcast\_info}'' \\ &\text{Broadcast\_info}' \\ &\quad = (\text{Broadcast\_info}, \text{is\_monitored}, \text{Yes}) \\ &\text{Broadcast\_info}'' \\ &\quad = (\text{Broadcast\_info}, \text{is\_intercepted}, \text{Yes}). \end{aligned} \tag{6}$$

Behavior is not independent and irrelevant; there is always a relation between the outputs of one behavior corresponding to the inputs of another behavior. One of the most common relationships is data dependence; that is, the execution input of the later behavior needs the execution output of the previous behavior. Another common relationship is control dependence; although two behaviors have no direct data flow relationship, the execution of the latter behaviors needs the execution of the previous one as a trigger condition. In former example, the relation between *monitor* and *intercept* is data dependence relation and can be described as follows:

$$\begin{aligned} \text{Output}(\varphi_{\text{monitor}}) &\longrightarrow \text{Input}(\varphi_{\text{intercept}}) \\ \text{Broadcast\_info} &\longrightarrow \text{Broadcast\_info}' \\ &\longrightarrow \text{Broadcast\_info}'' \end{aligned} \tag{7}$$

Definition 4. Software purpose is the final state of all input objects that act in an intention, formally represented as

$$\begin{aligned} \text{Goal} = &(\text{object}_1, \text{attribute}, \text{attri\_value}) \times \cdots \\ &\times (\text{object}_n, \text{attribute}, \text{attri\_value}). \end{aligned} \tag{8}$$

In purpose representation, all input objects, $\text{object}_1 \cdots \text{object}_n$, represent the objects involved in the intention. For example, after the extraction of sensitive behaviors in a malware and having a formal semantic representation, we use the ontology reasoning engine to automate the reasoning. We can eventually get the description of the relationship between behaviors and the final states of all the objects. For instance, we use "(SMS, position, "1062568")" to illustrate that the SMS is sent to "1062568"; (URL, is_used, Yes) represents the fact that the URL is used in the related function, and (DeviceId, is_encrypt, Yes) illustrates that the DeviceId has been encrypted.

3.3. Proof of Model. If intention is viewed as a system [22], the basic component of intention is therefore the behavior. After the inputs and outputs involved in these behaviors are defined, a series of basic mapping relationships are obtained which reflects the influence on external object of the basic behavior. The overall nature of an intention system can be determined by a set of mappings between a set of inputs and a set of outputs. Lemmas 5 and 6 are the basic mathematical theory of our behavior deduction.

Lemma 5 (function combination). *The union of mapping set φ_i is a mapping $\varphi : A \to B$:*

$$\varphi = \bigcup_{i=1}^{n} \varphi_i,$$

$$A = \bigcup_{i=1}^{n} A_i,$$

$$B = \bigcup_{i=1}^{n} B_i$$
$$\varphi_i : A_i \to B_i, \quad 1 \le i \le n \tag{9}$$

when $(\forall A_i)(\forall A_j)(A_i \subseteq A \wedge A_j \subseteq A \wedge A_i \neq A_j \to A_i \cap A_j = \varnothing)$.

Lemma 6 (function compound). *Two-tuple φ consisted of a set of mappings Φ and its compound relation ξ° is a mapping $\varphi : \mathrm{dom}(\varphi_1) = \mathrm{ran}(\varphi_n)$:*

$$\begin{aligned} \varphi &= (\Phi, \xi^\circ) \\ \Phi &= \{\varphi_i \mid \varphi_i : A_i \to B_i,\ A_i = \mathrm{dom}(\varphi_i),\ B_i \\ &\quad = \mathrm{ran}(\varphi_i),\ 1 \le i \le n\} \\ \xi^\circ &= \{\varphi_{i+1} \circ \varphi_i \mid \varphi_i,\ \varphi_{i+1} \in \Phi,\ 1 \le i \le n-1\} \end{aligned} \tag{10}$$

when $(\forall i)((1 \le i \le n-1) \to \mathrm{ran}(\varphi_i) \subseteq \mathrm{dom}(\varphi_{i+1}))$.

According to Lemmas 5 and 6, we proved Corollaries 7 and 8.

Corollary 7 (compound intention deduction). *$\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ represents a set of behaviors that are involved in an intention. If the relation between these behaviors is $Output(\varphi_i) = Input(\varphi_{i+1})$, $1 \le i \le n-1$, the output $Output(\varphi_n)$ of behavior φ_n is therefore the representation of final goal; behavior sequence is denoted as $\xi_i = (\varphi_1, \varphi_2, \ldots, \varphi_n)$.*

Proof. $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ are a set of behaviors. By the definition of φ, behavior satisfies the mapping relation between objects and can be regarded as function mapping. φ_i and φ_{i+1} satisfy the former output corresponding to the latter input; that is, the range of φ_i is equal to the domain of φ_{i+1}, which satisfies $(\forall i)((1 \le i \le n-1) \to \mathrm{ran}(\varphi_i) \subseteq \mathrm{dom}(\varphi_{i+1}))$. According to Lemma 6, Corollary is proved. □

Corollary 8 (combination-compound intention deduction). *$\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ represents a set of behaviors that are involved in an intention. If there is an input to output relation $Output(\varphi_m) = Input(\varphi_u)$, $1 \le m, u \le n$, and parallel relationship $Output(\varphi_i) \cup Output(\varphi_j) = Input(\varphi_k)$, $1 \le i, j, k \le n$, at the same time, then the output $Output(\varphi_u) \cup Output(\varphi_k)$ of φ_u and φ_k is the final goal. Behavior sequences are denoted as $\xi_i = (\Phi, R)$. Φ is behavior set; R is the relationship between these behaviors.*

Proof. $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ is known as a set of behaviors. By the definition of φ, behavior satisfies the mapping relation between objects and can be regarded as function mapping. φ_m and φ_u satisfy the former's output corresponding to the latter's input; that is, the range of φ_m is equal to the domain of φ_u. $\mathrm{dom}(\varphi_u)$ is the compound output according to Lemma 6. φ_i and φ_j satisfy both of their outputs as the input of φ_k; because the intersection of the inputs of φ_i and φ_j is empty $(\mathrm{ran}(\varphi_i) \cup \mathrm{ran}(\varphi_j) = \varnothing)$, they satisfy Lemma 5. φ_i and φ_j satisfy combination relation, which could be combined into a new mapping:

$$\varphi' : \mathrm{ran}(\varphi_i) \cup \mathrm{ran}(\varphi_j) \longrightarrow \mathrm{dom}(\varphi_i) \cup \mathrm{dom}(\varphi_j). \tag{11}$$

Then, φ' and φ_k satisfy the condition of Lemma 6; $\mathrm{dom}(\varphi_k)$ is the compound output. Corollary is proved according to Lemma 6. □

4. Ontology Inference System

This chapter elaborates the construction process of ontology inference system. Section 1 gives the construction specification of ontology model [10]. Section 2 gives SWRL rules used in inference engine. Section 3 is the description of mapping methods from Data Source to Fact Base. Section 4 elaborates the framework of inference system.

4.1. Ontology Model. The reasons for using ontology in our inference system are as follows: (1) The conceptual system based on ontology is computable, and the automatic deduction of intentions can be achieved by the computing between concepts. The system has reduced the workload of code writing accordingly. (2) The extensibility of ontology enables us to extend the conceptual model at any time according to the emergence of new features of malware. Once new malware knowledge appears we only need to modify the ontology model based on the knowledge. In this way, we have reduced the amount of code update and maintenance. (3) The ontology model of the elements and their relationships in malware behavior intention domain can standardize domain knowledge and make it shareable [10]. It can provide a standardized representation for malware intention.

The ontology model of malware behavior intention uses the following knowledge: the definition and classification of behavior and behavior object, intention model, and Corollaries 7 and 8. The concepts and the relation are shown in Figure 1. The definitions of each concept are given in Section 3. We use Pellet engine reasoning to verify the consistence of ontology.

The definitions of object attributes and data attributes are illustrated in Table 2.

4.2. SWRL Inference Rules. This section uses the knowledge of Corollaries 7 and 8 and Definition 4 in Section 3. Before writing inference rules, we need to define the format of basic facts. The fact is a composition description of basic behavior. There are two categories of basic facts (in our research): B1: the behavior with single input and single output; B2: the behavior with double input and double output. See Box 1.

(1) Inference Rules of the Relationship between Behaviors

Premise 1. Any two B1 behaviors are B11 and B12; B11(in) represents the input of B11, and B11(out) represents its output. If any two B1 behaviors B11 and B12 have the relation of B11's

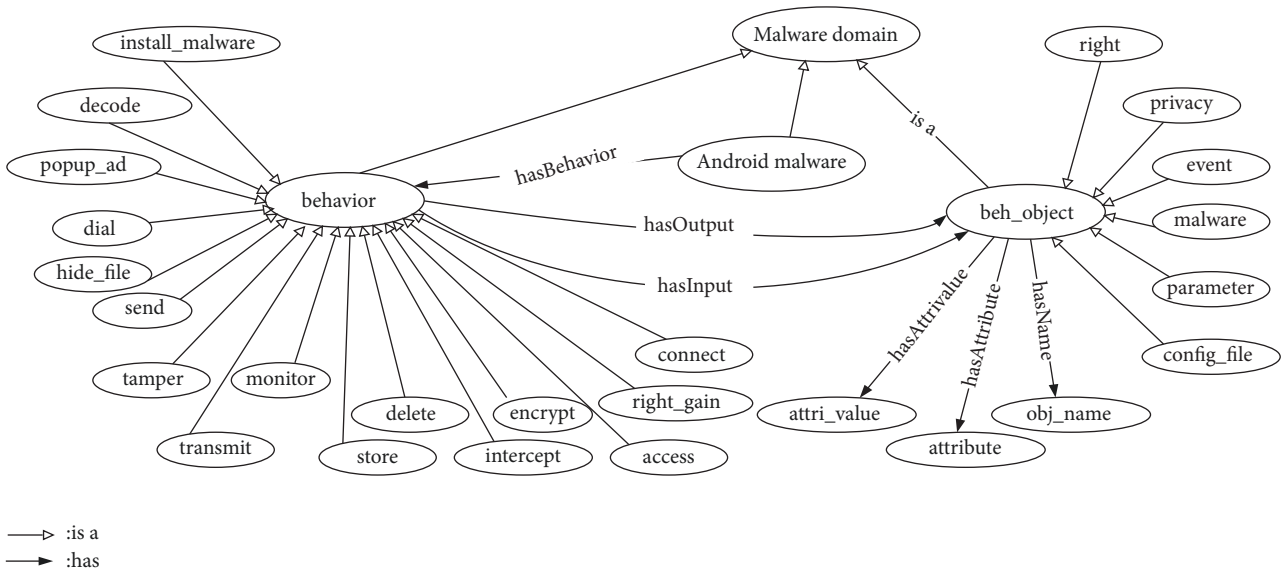

Figure 1: Semantic ontology of behavior intention.

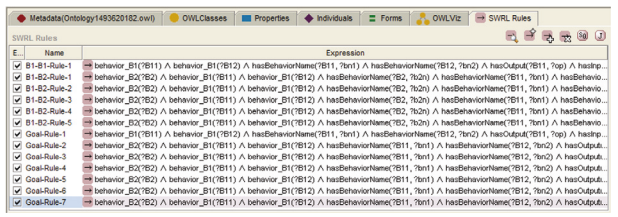

Figure 2: SWRL inference rules of the relationship between behaviors and final goal.

output object corresponding to B12's object input, Rule-1 is the inference rule corresponding to this condition. See Box 1.

Premise 2. Any two B1 behaviors are B11 and B12; one B2 behavior is B2. B11(in) represents the input to B11, and B11(out) represents its output. B2(1in), B2(2in), B2(1out), and B2(2out) represent the input and output of B2, respectively.

If the output's union of any two B1 behaviors is the input of a certain B2 behavior and the input intersection of these two B1 behaviors is empty, then B11 and B12 have a compound relation with the B2 behavior, respectively. There is also a combination relation between B11 and B12. The rule is B1-B2-Rule-1 shown in Figure 2 and the rest of the rules are similar.

(2) Inference Rules of the Final Goal. On the basis of behavior's inference rules and Definition 4, the inference rules of final goal are summarized as in Figure 2, such as Goal-Rule-1. The outputs are the descriptions of object's final state. Due to the limited space, other rules are no longer presented.

4.3. Extraction of Behavior Facts. The basic facts include B1 and B2 behavior, and the behavior's elements which should be extracted from Data Source are as follows:

(1) Behavior (name): extracting the behavior description in a program according to the mapping between the behaviors defined in our sensitive API database and sensitive APIs (or code segment) in program.

(2) Input (behavior object): we determine objects description based on the parameters of sensitive APIs and the official document definition.

(3) Output (behavior object): object after a behavior operates on it, its object name, and attribute will not change while its attribute value will be affected. Therefore, we can determine the changes in attribute values based on this behavior's definition.

The mapping from Data Source to behavior facts is divided into two stages: the first stage is behavior recognition; the second includes object identification and relation analysis between objects. Detailed process is as follows.

First Stage. Use reverse tool to decompile these malware samples and then generate its call graph (CG) and control flow graph (CFG). The leaf nodes of the call graph are traversed to find sensitive API and identify the corresponding behavior according to the mapping relation between behavior and sensitive API calls. Partial mapping relations between behaviors and sensitive API calls and object descriptions are shown in Table 3.

Second Stage. For each identified behavior, the behavior's object is identified according to the usage of parameter in

Table 2: Specification of object attributes and data attributes.

Attribute name	Specification
hasCombinationwith	Behaviors have combination relation
hasCompoundwith	Behaviors have compound relation
hasInput(x, y)	The behavior's single input is "y"
hasOutput(x, z)	The behavior's single output is "z"
hasFirstInput(x, y1)	The first input of double input is "y1"
hasSecondInput(x, y2)	The second input of double input is "y2"
hasFirstOutput(x, z1)	The first output of double input is "z1"
hasSecondOutput(x, z2)	The second output of double input is "z2"
hasBehavior(x, y)	The behavior belonging to malware is "y"
hasObjectName(x, y)	The name of the object is "y"
hasAttribute(x, y)	The attributes of an object are "y"
hasAttributeValue(x, y)	The object's attributes value is "y"

Rule-1: behavior_B1(?B11) ∧ behavior_B1(?B12) ∧ has BehaviorName(?B11, ?bn1)∧hasBehaviorName (?B12, ?bn2) ∧ hasOutput(?B11,?op) ∧ hasInput(?B12,?ip)∧ hasObject name(?op, ?on1) ∧ hasObjectname(?ip, ?on2)∧swrlb:equal (?on1,?on2) ∧ hasAttribute(?op, ?att1) ∧ has Attribute (?ip, ?att2) ∧ hasValue(?att1, ?attri1) ∧ hasValue (?att2, ?attri2) ∧ swrlb:equal (?attri1, ?attri2) ∧ hasAttribute Value(?op,?attv1) ∧ hasAttributeValue (?ip, ?attv2) ∧ hasValue (?attv1, ?attriv1) ∧ hasValue(?attv2,?attriv2) ∧ swrlb:equal (?attriv1,?attriv2) → hasCompoundwith(?bn1, ?bn2).

Box 1

sensitive API (two categories: the first obtains behavior object based on the class object and the definition of API, such as getDeviceId(); the second obtains behavior object based on the parameter usage of API and the definition of API, such as Runtime.exec (RootExploit.file)). Data dependence between behaviors is analyzed using FlowDroid [4].

4.4. Framework of Inference System. The framework of intention inference system is shown in Figure 3.

We use the knowledge of intention definition and Corollaries 7 and 8 to construct SWRL rules (the rules are represented in ontology language). Extracting Data Source from Android applications uses reverse engineering technology [15]. The extraction methods of facts are illustrated in Section 4.3. Jess inference engine completes the reasoning process using the Facts Base and SWRL rules and gives the inference output (the description of malware intention) finally.

4.5. Motivation Example. Take Zitmo [16] as an example to show the facts extracted from these samples, as shown in Table 4. Among these extracted facts, *access* and *transmit* are B2 behavior; B2's double input indicates that the execution of the behavior needs to meet the condition of the two key inputs.

After the rules and facts are imported to Jess engine, the inference results are exported; they are the formalized descriptions of intentions of each malicious sample. The results of Zitmo's behavior relations are shown in Figure 4. Other results of intention reasoning are analyzed separately and show reasonable efficiency.

We extract behavior relations and goal representation from Figure 4, which are shown in Table 5.

Zitmo's intention includes behavior sequence (consisting of behavior set and behavior relation) and the representation of goal (output object description). To visually demonstrate the reasoning results, we display them in Figure 5 graphically. In Figure 5, rectangle represents the behavior, ellipse represents the input or output object of the behavior, and arrow indicates the relationship between object and behavior. The ellipse with a black font represents the raw input of objects involved in the intention, respectively, as broadcast information, text messages, and URL addresses. These final states are as follows: broadcast message is monitored and intercepted, the contents of the messages are transmitted to remote server, and the URL is used as the destination address of the transmission. They represent the goal of the intention together.

5. Evaluation

We have implemented the ontology inference system described in Section 4.4 in a prototype inference system for identifying and describing malware intention in behavior

Table 3: Partial mappings between behavior and sensitive APIs.

Behavior	Behavior object	sensitiveAPI
right_gain	Root permission	Runtime.exec()
access	Device ID	getDeviceId()
	Carrier name	getNetworkOperatorName()
	Phone position	getCellLocation()
	Short Message	createFromPdu()
send	Number and contents	sendTextMessage()
intercept	Broadcast information	abortBroadcast()
connect	URL-parameter	URLConnection.connect()
transmit	Parameter	execute()
encrypt	Parameter	setEntity()
store	Parameter	writeRec()

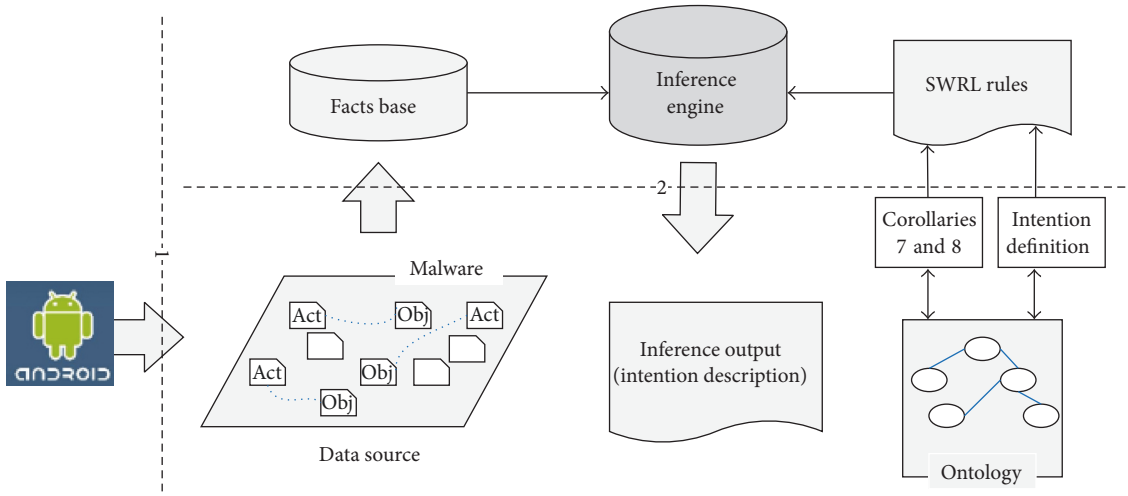

Figure 3: The framework of intention inference system.

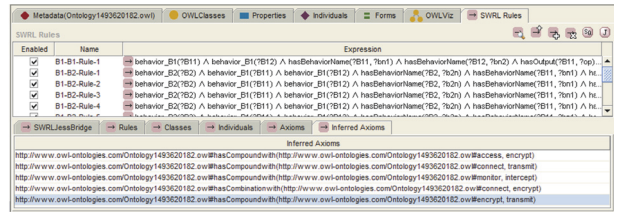

Figure 4: Inference results of Zitmo's behavior relations.

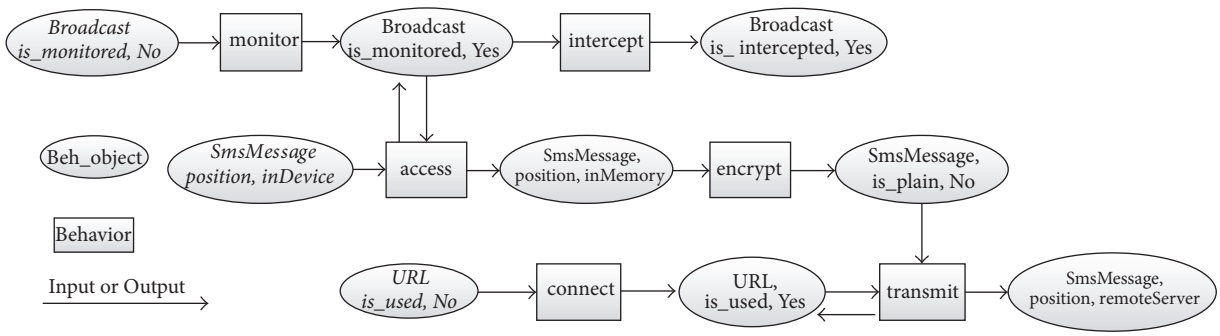

Figure 5: Visualization of Zitmo's inference results.

Table 4: Behavior facts extracted from Zitmo.

Behavior	Code segment or function	Behavior object
monitor	onReceive()	Broadcast
intercept	abortBroadcast()	Broadcast'
access	createFrompdu(), getMessageBody(), getOriginatingAddress()	Broadcast', SmsMessage
connect	new HttpPost(URL)	URL
encrypt	setEntity()	SmsMessage
transmit	execute()	URL', SmsMessage

Table 5: The inference results of Zitmo.

Behavior relationship	Final goal representation
hasCompoundwith(monitor, intercept)	(Broadcast, is_monitored, Yes), (Broadcast, is_intercepted, Yes)
hasCompoundwith(monitor, access)	(SmsMessage, position, inMemory)
hasCompoundwith(access, encrypt)	(SmsMessage, is_plain, No)
hasCombinationwith(connect, encrypt)	No
hasCompoundwith(encrypt, transmit)	(URL, is_used, Yes)
hasCompoundwith(connect, transmit)	(SmsMessage, position, remoteServer)

facts. In this section, we evaluate the following aspects: First, we verify whether the sensitive API library we build can cover the behavior facts that exist in malware samples and give the coverage rate analysis. Second, to verify the effectiveness of artificial extraction (semiautomated) behavior facts, we compare our manual analysis results with the results generated by automated tool CopperDroid [19]. Thirdly, the effectiveness and reasoning performance of the ontology inference system are shown. Finally, the correctness, readability, and effectiveness of the results of intention reasoning are tackled.

We have selected 75 typical malware samples (1 real-world ransomware sample and 9 real-world samples from Genome's 1260 samples and 65 DroidBench samples) for evaluation, which are shown in Table 6. Among these 10 real-world samples, 9 samples are collected from Android malware database established by Zhou and Jiang [23] in North Carolina State University. We use these real-world samples because they are typical and representative of their family. Other samples in Genome are in same family, so the vulnerabilities and threats in the program are similar. For efficiency consideration, we do not add the remaining samples to experimental sample set. DroidBench [4] samples are designed to assess the correctness of static analyses on Android apps. We use these samples as the ground truths because they have open-sourced programs with clear semantics [20]. Thus we can make an adequate analysis of these samples. In fact, static analysis technique in general often lacks capability of extracting runtime behaviors and can be evaded accordingly, but it has incomparable advantages over dynamic technology. Nevertheless, this paper focuses on the research of software intention reduction; that is, the emphasis is to study the extraction and semantic mapping methods of malware's behavior, the extraction of relationship between behaviors, and the representation of malware's final goal.

5.1. Effectiveness of Behavior Facts Extraction. In general, we have discovered 102 significant behaviors involving 115 sensitive APIs, via sensitive API mining in Genome's Android apps and the summary of related literature [15, 19–21]. We also refer to Android's official API document and based on the information collected a database of sensitive API behavior is established. These APIs are controlled by 30 Android sensitive privileges and have detailed descriptions in official documents.

To evaluate the effectiveness and validity of behavior facts extracting, we compare the description of our behavior facts extraction and CopperDroid's [19] behavior description. CopperDroid is an automatic VMI-based dynamic analysis system to reconstruct the behaviors of Android malware; the novelty of CopperDroid lies in its agnostic approach to identify interesting OS- and high-level Android-specific behaviors. We perform a user study on CopperDroid platform and our extraction methods; the goal is twofold. First, we give a comparative analysis on the behavior coverage rate of these samples. Second, we hope to know whether the behavior description generated by our methods is readable to average audience. To this end, we compare the behavior description of our methods and CopperDroid's Public Reports [24]. We have collected 150 copies of CopperDroid's public malware sample analysis report and make a statistical analysis. As

Table 6: Malware samples and their behavior information.

Sample family	Number	Ours (Copper)	Major behavior	Intention classification
Zitmo	1	6 (2)	monitor, intercept, access, connect, encrypt, transmit	Privacy stealing
GoldDream	1	9 (4)	monitor, access, store, connect, transmit	Privacy stealing
DroidDream	1	5 (3)	right_gain, access, connect, transmit	Privacy stealing
DroidDeluxe	1	3 (2)	monitor, right_gain, access, connect, transmit	Privacy stealing
HippoSMS	1	4 (2)	send, monitor, access, intercept	Tariff consumption
Geinimi	1	6 (3)	remote_control, send, connect, transmit	Tariff consumption
RogueSPPush	1	6 (2)	send, monitor, access, intercept, delete	Malicious chargeback
GGTracker	1	9 (4)	access, connect, transmit, store, encrypt, monitor, send, intercept	Malicious chargeback
DroidKungFu-Update	1	4 (2)	connect, transmit, install_mal, popup	Malware propagation
Love buckle word	1	4 (1)	popup, tamper, connect, transmit, right_gain	Extortion user
Aliasing	1	2 (2)	access, send	Privacy leak
AndroidSpecific	9	3 (2)	access, logging, send	Privacy leak
ArraysAndLists	7	2 (2)	access, send	Privacy leak
Callbacks	4	2 (2)	access, send	Privacy leak
EmulatorDetection	3	3 (2)	access, logging, send	Privacy leak
FieldAndObjectSensitivity	3	3 (2)	access, logging, send	Privacy leak
GeneralJava	14	3 (2)	access, logging, send	Privacy leak
ImplicitFlows	4	2 (2)	access, logging	Privacy leak
InterAppCommunication	3	2 (2)	access, send	Privacy leak
Lifecycle	11	4 (2)	access, connect, send, logging	Privacy leak
Reflection	4	2 (2)	access, send	Privacy leak
Threading	2	3 (2)	access, logging, send	Privacy leak

Table 7: Behavior semantics between ours and CopperDroid.

Behavior class	Objects	Copper	Ours	Similar
Access personal info	SMS phone accounts location	4	18	90%
FS access	Open Write	2	2	100%
Network access	Generic HTTP DNS	3	3	100%
Send SMS	SMS	1	1	100%
Execute external file	Generic Priv. Esc Shell Inst. APK	6	8	75%
Encrypt info	File info	0	2	0%
Intercept SMS	SMS	0	2	0%
Store info	Privacy info	0	1	0%

Table 8: Intention generation results for samples.

Sample set	False status	Missing desc	Correct	$T\#$
Genome	1	1	8	10
DroidBench	4	2	59	65
Total	5	3	67	75

shown in Table 6, the third column (ours/Copper) indicates the number of behaviors' extraction using our methods and the number of behaviors extracted by CopperDroid for the same sample. In terms of the number of extracted behaviors, our extraction mechanism (87) is 77.6% more than CopperDroid's behavior number (49).

We further study whether the behavior description generated by our methods is readable to average audience. As behavior facts extraction is the basis of this study, we thus produce 75 behavior descriptions reports for these 75 samples. The result, illustrated in Table 7, depicts that the descriptions derived from ours are richer than CopperDroid, while the ratio of behavior similarity ranges from 75% to 100%. CopperDroid failed to extract some of the behaviors we extract, such as encrypt, intercept, and store personal info. Thus we argue that the behaviors we extract can reflect the common program semantics and programming conventions effectively.

Notice that though it may take artificial effort to generate behavior facts description, behavior extraction is a technology that has already been realized. We refer to the work of predecessors extracting malware behaviors though automatic extraction technology is not so mature. The focus of this work is to propose a framework for the description and deduction of behavioral intentions. We argue that any analysis tools, both static and dynamic, can be utilized in our framework to achieve the described malware intention.

5.2. Validity of Ontology Inference System. Next, we evaluate the validity of ontology inference system. Ontology inference experiment reveals that the workload of manual writing code reduced significantly and Pellet engine reasoning verified that the ontology is complete and consistent. We evaluate the runtime performance and ontology validity for 75 samples; preparation for behavior facts (add ontology instance) dominates the runtime, while the intention results generation is usually fairly fast (under 200 milliseconds). The average inference runtime is 2 seconds, while the analysis for a majority (85%) of apps can be completed within 5 seconds. The validation of the reasoning results indicates that 89.3% of the sample's reasoning results are correct and the intention results of graphical display show high readability; Table 9 shows the readability ratings of 75 apps.

5.3. Correctness, Readability, and Effectiveness

Correctness. To evaluate the correctness, we produce intention descriptions for 75 malware sample apps (10 typical real-world samples and 65 DroidBench samples) using our behavior facts extraction algorithm and ontology inference engine. First, we use artificial methods to simulate automation tools to extract the behavior facts in these samples. The extraction mechanism is to construct the environment dependence graph first and then use the methods in Section 4.3 to map the sensitive APIs to behavior facts. Second, the extracted behavior facts are instantiated in the tools provided by protégé 3.4.7, and then the instances that are obviously wrong are filtered. Finally, we use Jess inference engine for reasoning the behavior facts, and the SWRL rules are imported to inference engine at the same time. The inference results give the output of behavior sequence and final goal representation (see the definition of malware intention); finally, the framework is shown in Figure 3.

Results. Table 8 presents the experimental results, which show that our inference system achieves a true positive rate of 89.3%. We must reiterate that the basis of our reasoning is the sensitive API database we built can fully map sensitive behaviors in malware. Although it is not perfect, it can be

Table 9: Precision results for Appscan and intention reports.

Judgment	Appscan report			Intention description		
	geno	Driod	All	geno	Droid	All
(1) Ineffective	5	0	5	1	2	3
(2) Uncertain	1	65	66	1	4	5
(3) Effective	4	0	4	8	59	67
Total	10	65	75	10	65	75

fully covered in the 75 selected samples. Inference system misses intention descriptions due to two major reasons: (1) Artificial behavior facts extraction lacks accuracy. We rely on our environment dependence graph and extraction algorithm to perform simulation analysis. However, it is not precise enough to handle the data flow propagation that cannot be solved by existing static analysis technology (such as exception handler code, reflective calls). (2) The program logic in real-world malware is more complex and with semantic ambiguity, and there are too many independent sensitive behaviors (in the form of user defined functions). This is not the result of the lack of theory and only technical problem.

In fact, static analysis in general lacks the capability of extracting runtime behaviors and can be evaded accordingly. Nevertheless, we argue that our focus is to construct a framework which produces description and derivation model of malware intention.

Readability and Effectiveness. To evaluate the readability and effectiveness of inference output (Intention), we perform a user study on our malware intention inference results. The goal is twofold. First, we hope to know whether the generated intentions are readable to average audience. Second, we expect to see whether our intention descriptions can actually help users avoid risky apps.

Methodology. 360 microscopes' Appscan platform [25] is an online security scanning system which focuses on the security scan of Android apps. The system will provide a scan report after the scanning and describe possible malicious intent of the app to users. We uploaded all the samples to 360 microscopes' Appscan platform and got 75 security reports of these samples. We also collate ontology inference results and show them in the form of formalization and graphics; 75 samples' intention reports are produced. We use both 75 Appscan reports and 75 intention reports to measure user reaction and the number of vulnerabilities provided by 360 microscopes' Appscan platform is compared with the intention description. Furthermore, due to the objective evaluation consideration, we perform the user study based on the official intents descriptions of 75 apps. We use these official descriptions as a reference standard.

We have recruited participant directly from our laboratory group and we require participant that must be smartphone users. We also make sure that participants have experiences with Android malware analysis and understand basic smartphone events, such as "intercept SMS" or "access GPS location." We provide a rating for each sample report (Appscan and intention) with respect to its effectiveness. The rating ranges from 1 to 3, where 1 means ineffective, 2 means uncertain, and 3 means effective, as shown in Table 9.

Results and Implications. Eventually, the results are shown in Table 9, 360 microscopes' Appscan platform provides a coarser grained description, and users can only judge the true intents of 4 samples according to the Appscan report. While users can judge the malice of 67 samples using intention description report. This indicates our intention report is readable, even compared to Appscan report created by 360 microscopes' Appscan platform. The results also reveal that the readability and granularity of intention description are relatively important while Appscan-generated ones sometimes confuse users; 66 samples cannot be judged correctly through Appscan report. In a further investigation, we notice that Appscan report provides a coarse intent description of the sample, while our security centered intention report provides a fairly fine-grained display of intention. We believe that this can be further utilized during threat analysis.

Threat Analysis. Analysts are able to understand the relationships between malware behaviors (including trigger process of behaviors, data transfer relation, and the change in object characteristics) according to the intention description. This is of great practical significance for users to understand the implementation mechanism of malware, evaluate its possible losses, and provide preventive solutions.

6. Conclusion

In this paper, a novel description and derivation model of Android malware intention based on the theory of intention and malware reverse engineering is proposed to restore the intention of malware and automate the process of intention reasoning process. In order to standardize the new conceptual system and automate the intent derivation, we create ontology of malware intention. As ontology is computable, we import SWRL rules represented in ontology language and Fact Base to Jess engine, and Jess's outputs are the description of malware intention. We evaluate our system using 75 typical malware samples (5 kinds). Experiments show that our inference system is capable of achieving the desired results of intention description, which proved its feasibility and effectiveness. In addition, by using existing reverse technology and data flow analysis tools we can extract behavioral facts effectively.

Methods of mapping from Data Source to facts are divided into two forms: artificial extraction and automatic extraction. We mainly use the methods of artificial facts extraction. There are many opportunities for future works, such as the implementation of automatic facts extraction methods and the automatic visualization of the reasoning results.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was partly supported by Beijing Key Laboratory of Internet Culture and Digital Dissemination Research Project (no. ICDDXN001), National Key Technology Research and Development Program of the Ministry of Science and Technology of China (no. 2015BAK12B03-03), Special Project of Central Government to Guide the Development of Local Science and Technology (no. Z171100004717002), and National Natural Science Foundation of China (nos. 61370065, 61502040).

References

[1] P. Guojun, L. Jingwen, and S. Runkang, "Research and Progress of Android malware detection," *Journal of Wuhan University (Science Edition)*, vol. 61, no. 1, pp. 21–33, 2015.

[2] A. Kharraz, S. Arshad, C. Mulliner et al., "A Large-Scale, Automated Approach to Detecting," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, pp. 757–772, USENIX Association, 2016.

[3] M.-Y. Su and K.-T. Fung, "Detection of android malware by static analysis on permissions and sensitive functions," in *Proceedings of the 8th International Conference on Ubiquitous and Future Networks, ICUFN 2016*, pp. 873–875, Austria, July 2016.

[4] S. Arzt, S. Rasthofer, C. Fritz et al., "FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[5] E. Bratman Michael, "What is intention?" in *Intentions in communication*, P. R. Cohen, Ed., pp. 15–32, the MIT Press, 1990.

[6] J. Shirley and D. Evans, "The user is not the enemy: Fighting malware by tracking user intentions," in *Proceedings of the New Security Paradigms Workshop 2008, NSPW '08*, pp. 33–45, USA, September 2008.

[7] C.-T. Jung, C.-H. Sun, and M. Yuan, "An ontology-enabled framework for a geospatial problem-solving environment," *Computers, Environment and Urban Systems*, vol. 38, no. 1, pp. 45–57, 2013.

[8] I. Horrocks, F. P. Patel-Schneider, H. Boley et al., *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, W3C Member Submission, 2004.

[9] J. Bak, C. Jedrzejek, and M. Falkowski, "Usage of the jess engine, rules and ontology to query a relational database," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Preface*, vol. 5858, pp. 216–230, 2009.

[10] X.-Y. Du, M. Li, and S. Wang, "Survey on ontology learning research," *Ruan Jian Xue Bao/Journal of Software*, vol. 17, no. 9, pp. 1837–1847, 2006.

[11] S. Cheng, S. Luo, Z. Li et al., "Static Detection of Dangerous Behaviors in Android Apps," in *Proceedings of the the 5th international symposium on cyberspace safety and security*, Springer International Publishing, 2013.

[12] IDA pro. http://www.hex-rays.com/products/ida.

[13] J. Jian and X. Qing, "On case auto generation using reverse analysis for Android malware," *Journal of HeFei University of Technology (Natural Science Edition)*, vol. 39, no. 4, pp. 466–470, 2016.

[14] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "AppContext: differentiating malicious and benign mobile app behaviors using context," in *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE '15)*, vol. 1, pp. 303–313, IEEE, May 2015.

[15] Y. Li, T. Shen, X. Sun et al., "Detection, Classification and Characterization of Android Malware Using API Data Dependency," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, vol. 164, pp. 23–40, Springer International Publishing, 2015.

[16] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS '14)*, pp. 1105–1116, ACM, Scottsdale, Ariz, USA, November 2014.

[17] P. Wu, H. Changzhen, and Y. Shuping, "A dynamic intrusive intention recognition method based on timed automata," *Journal of Computer Research and Development*, vol. 48, no. 7, pp. 1288–1297, 2011.

[18] J.-H. Han, S.-J. Lee, and J.-H. Kim, "Behavior Hierarchy-Based Affordance Map for Recognition of Human Intention and Its Application to Human-Robot Interaction," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 5, pp. 708–722, 2016.

[19] K. Tam, S. J. Khan, A. Fattori et al., "CopperDroid: automatic reconstruction of android malware behaviors," in *Proceedings of the Network and Distributed System Security Symposium*, San Diego, Calif, USA, 2015.

[20] M. Zhang and H. Yin, "Automatic Generation of Security-Centric Descriptions for Android Apps," in *Android Application Security*, Springer International Publishing, 2016.

[21] Z. Wang, C. Li, Z. Yuan, Y. Guan, and Y. Xue, "DroidChain: A novel Android malware detection method based on behavior chains," *Pervasive and Mobile Computing*, vol. 32, pp. 3–14, 2016.

[22] X. Guozhi, *System science and Engineering Research*, Shanghai science and Technology Education Press, 2001.

[23] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, pp. 95–109, IEEE, 2012.

[24] http://copperdroid.isg.rhul.ac.uk/copperdroid/index.php.

[25] http://appscan.360.cn/.