



Ontology-based knowledge representation for malware individuals and families

Yuxin Ding*, Rui Wu, Xiao Zhang

Harbin Institute of Technology (Shenzhen), Shenzhen University Town, Shenzhen, China

ARTICLE INFO

Article history:

Received 20 January 2019

Revised 21 July 2019

Accepted 23 July 2019

Available online 24 July 2019

Keywords:

Ontology

Malware

Dynamic behavior

Malware detection

Knowledge base

ABSTRACT

Malware consists of a large numbers of malware families and individuals, and each individual has complex behaviors. So knowledge base is urgently needed to process and store such a huge amount of information. In present the traditional signature-based database cannot represent the behavioral semantics of malicious code. Therefore, people cannot know what malware will do on a computer system. To solve this issue, we apply ontology technique into the malware domain, and propose the method for constructing malware knowledge base. We design the concept classes and object properties of malware, and propose the method for representing semantics of malware behavior. The data mining method, Apriori algorithm, is applied to extract the common behaviors of individuals belonging to the same family, and common behaviors are used to represent the knowledge of a malware family. The experimental results show that the data mining method can discover the common behaviors of the malware family, and the common behaviors mined can effectively classify the malware families.

© 2019 Elsevier Ltd. All rights reserved.

1. Introduction

At present, more and more devices are connected to Internet, such as smartphones and household appliances. With the increase of Internet connected devices, users are more likely to become targets of network attack. One of the major threats facing computer systems and their users today is malicious code (malware). Malware has complex behaviors, and can use different technologies to attack computer systems. Usually it can bypass the security mechanism, and install itself on the target host, and establish remote access.

People mainly use anti-virus software to detect malware. Most of these anti-virus tools use signature-based methods to detect malware (Filiol, 2006). A signature is a short string of bytes which is unique for each known malware. The representation of malware signature is simple, so it is very easy to build a large signature database for detecting malware. However, signatures have no semantics, and we cannot know what malware has done on a computer system. In addition, signature of malware can be easily modified. Malware writers can use obfuscation techniques to change signature, which can produce a large number of variants of a malware family. This results the signature-based methods fail to detect variants of known or previously unknown malware.

To solve this issue, behavior based detection methods are proposed. Malware behavior has different representations, for example, API sequences and opcode sequences all can be used to describe malware behavior. However, API sequences and opcode sequences cannot accurately represent malware behavior. People also cannot know what malware has done on the target machine. To obtain the exact behaviors of malware, it is necessary to run malware in the virtual environment and monitor the behavior of malware continuously. Malware is a super complex group. It contains many families, and each family is composed of a large number of individuals, and each individual shows complex behaviors. One problem we need to solve is how to represent and store such huge amount of information so that the machine can understand and process it automatically.

In this paper we study the representation of malware behavior and the construction method of the malware knowledge base. The purpose of this study is to develop an integrated knowledge base to represent and store behavior knowledge about malware individuals and families and help people analyze and detect malware. We introduce the ontology technology into the malware detection domain, and utilize ontology to describe malware behavior and construct the knowledge framework of malware, and use ontology reasoning technology to identify families of unknown malware.

* Corresponding author.

E-mail address: ding_yuxin@hotmail.com (Y. Ding).

2. Related works

In this paper, we use ontology technique to build malware knowledge base. So we only shows some research work related with ontology. Ontology can objectively describe things in the real world, so it is often used to describe the knowledge of a domain. Ontology has been applied in many fields such as knowledge engineering, artificial intelligence, web semantics and so on. In the field of information security, ontology has been used to describe the knowledge of malware.

Tafazzoli and Sadjadi (2008) proposed an ontology model to describe malware types and the relations between malwares. The ontology contains a main class named "Malact" including two subclasses, artifact and non-artifact. The class artifact represent malware class which includes eight malware classes. The class non-artifact contains the class spam. Malware properties include the description of four characteristics and their respective values: malware objectives, behavioral and technical features, malware architecture and target's placement, malware communication and management.

Martinez et al. (2010) proposed a cloud computing platform for malware detection. Files are submitted to the platform through a Web interface, and multiple engines are used to check a file as malicious or not. In the ontology model, the Semantic Web Rule Language is used to define detection rules. A SWRL rule includes an antecedent part to describe the body, and a consequent part as the head. They defined around 2224 attacks and malware signatures and 2210 prevention rules in the ontology model. Therefore, this ontology may be seen as a set of signatures to detect network intrusion and malicious code that are already known by antivirus engines. Wang et al. (2018) employed the support vector machine to evaluate the importance of different program features and used a multi-classifier system to identify malware. In the paper (Wang et al., 2019) the authors systematically surveyed the features constructed for detecting Android malware.

Huang et al. (2014) proposed a knowledge platform TWMAN that aims to provide a knowledge and rule base about malware samples. The TWMAN platform is composed of three layers, knowledge, communication and application. The TWMAN's ontology mainly contains four concept calsses: Malware_Impact_Target, Malware_Type, Malware Behavior, and Malware_Sample. While Malware_Impact_Target and Malware_Type are limited to network, registry, or file, and trojan, worm, or backdoor, respectively. André Grégio et al. proposed a Malicious Behavior Ontology that represents complex behaviors of suspicious executions, and they used inference rules to evaluate the associated threat level of malware.

Obrst et al. (2012) developed a Cyber ontology from an initial malware ontology. They described the potential ontology and standard that could be used to extend the Cyber ontology from its initially constrained malware focus. These resources contain Cyber and malware standards, schemas, and terminologies that directly contributed to the initial malware ontology effort. Their goal is to integrate data from different security-related sources and to reuse existing ontologies of the field, such as those describing attacks, vulnerabilities and malware. Michael et al. (2015) proposed an ontology model for a cyber security knowledge graph database, which is intended to design an organized schema that incorporates information from a large variety of structured and unstructured data sources, and contains all relevant concepts within the domain. Mundie and McIntire (2013) built an OWL-based malware analysis ontology for exchanging incident information, training staff, creating related courses etc. They built a malware analysis dictionary and taxonomy and combined them with a competency model to create an ontology-based competency framework.

Jasiul et al. (2014) also designed ontologies and detection rules to detect malware. They developed a tool named PRONTO, which

can collect system events about registry, process, file, and network from event logs and correlate these event using predefined malware behaviors modeled by Colored Petri networks. Shoaib and Farooq (2015) designed an email classifier to classify Spam and Ham (legitimate messages). In order to reduce false alarm rate, they designed an ontology model to describe users' interests regarding e-mail messages.

Due to limited processing power, storage capacity and battery power, it is difficult to distribute malware signatures files to mobile devices on time. To solve this issue, Chiang and Tsaor (2010) proposed an ontology-based behavioral analysis for mobile malware, and further provided information about mobile malware for end users to help them use their mobile phones securely. In their model malware behavior is classified according to the damage, type of threat, infection routes and spreading mechanisms. Wang et al. (2015) proposed a threat risk analysis model for mobile viruses. They explored a heuristic approach incorporating both malware behavior analysis and code analysis to create a virus behavior ontology. The proposed model overcomes the difficulties in identifying unknown threats and polymorphic viruses, using a VM-enabled environment. Navarro et al. (2018) proposed an ontology-based framework to model the relationships between application and system elements. They employed a machine-learning approach to analyze the complex network and identify characteristics shared by malware samples.

In addition, Ontology is also applied in network intrusion domain. Undercoffer et al. (2004) proposed an ontology specifying a model of computer attack using the DARPA Agent Markup Language+Ontology Inference Layer, a descriptive logic language. The ontology model focus on low level kernel attributes at the process, system and network levels, to serve as those taxonomic characteristics. More et al. (2012) improved the ontology proposed by Undercoffer et al. (2003). The model integrates these heterogeneous data sources and establishes a semantic-rich knowledge base to detect network threats/vulnerabilities.

Abdoli et al. (2010) created the ontology of DOS attack. To determine the consistency and accuracy of the designed ontology, they used Racer software and tested ontology through KDD CUP99 test data set. Hansman et al. (2005) proposed a four-dimensional taxonomy that provides a holistic taxonomy for dealing with inherent problems in the field of computer and network attacks. Based on proven concepts, Simmonds et al. (2004) proposed a framework for network security, and designed an ontology for network security attacks that showed the relationships between many standard classifications.

Different from our research, most of the above works focus on how to use ontology to define malicious behavior and detect malware. In our work, we study how to use ontology to represent the knowledge of malware individuals and families, and how to build the framework of malware knowledge base. Thus the knowledge in the database can be understood and processed automatically by machines. In this way, using the stored knowledge, we can not only decide which behaviors are malicious, but also infer the relationships among individuals and families.

3. Design of malware knowledge base

3.1. Ontology concepts for malware

Ontology is the science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality (Barry and Christopher, 2001). The ontology framework provides a consistent conceptual description of domain knowledge, which can share knowledge, facilitate knowledge reuse and reduce repetitive descriptions.

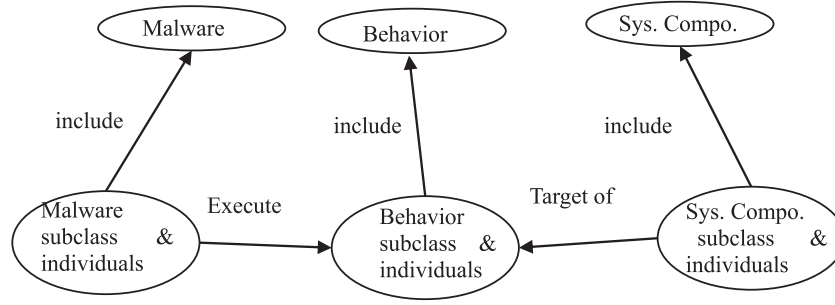


Fig. 1. Conceptual model of malware ontology.

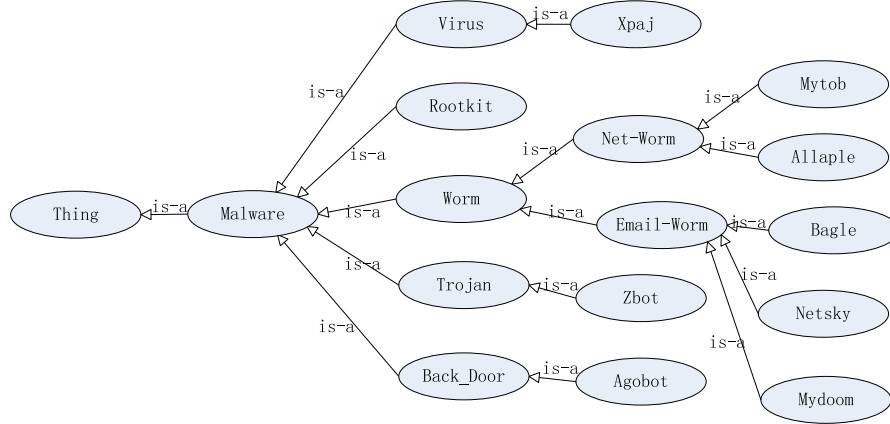


Fig. 2. Parts of the class structure of malware in the ontology model.

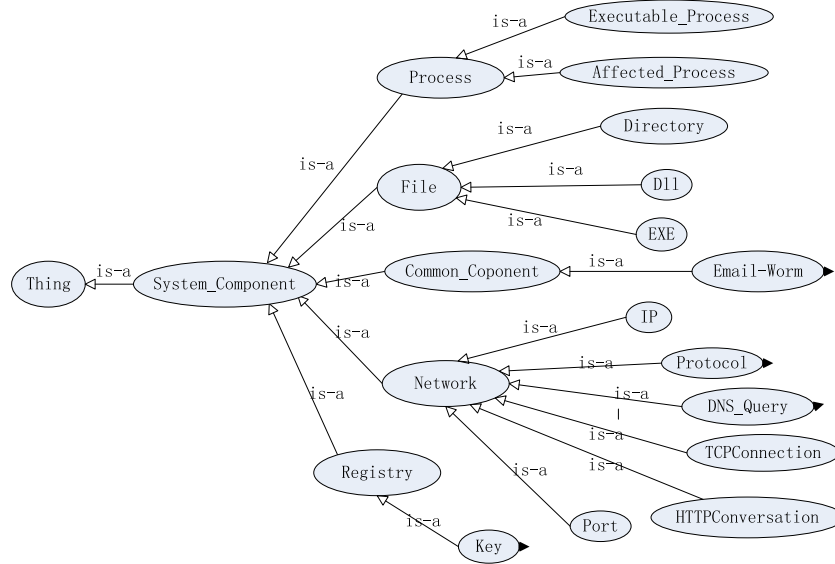


Fig. 3. Part of the class structure of System components in the ontology model.

Typically, ontology consists of classes, properties, relationships between classes and individuals. They describe important concepts (classes of objects) of domain and their properties. Ontologies are designed in a way that allows knowledge inference and reasoning, which are different from terminologies which are static structures used for knowledge reference. Malware ontology is a knowledge model of malware domain. It contains all of the relevant concepts related to malware behaviors, malware classes and individuals, computer system components.

The conceptual model of the proposed malware ontology is describe in Fig. 1, which presents an overview of the core classes,

and the relationships between classes and individuals. Malware, computer system component and behavior are three core classes of malware ontology. The malware class defines the classification architecture of malware, which includes all malware subclasses and individuals. The computer system component class defines the classification architecture of computer components, which includes all system component subclasses and individuals. The behavior class defines the classification architecture of malware behaviors, which includes different types of behaviors. Usually a malware is the executor of a behavior, and system components are the targets of behaviors. Therefore, behavior class describes the relationships

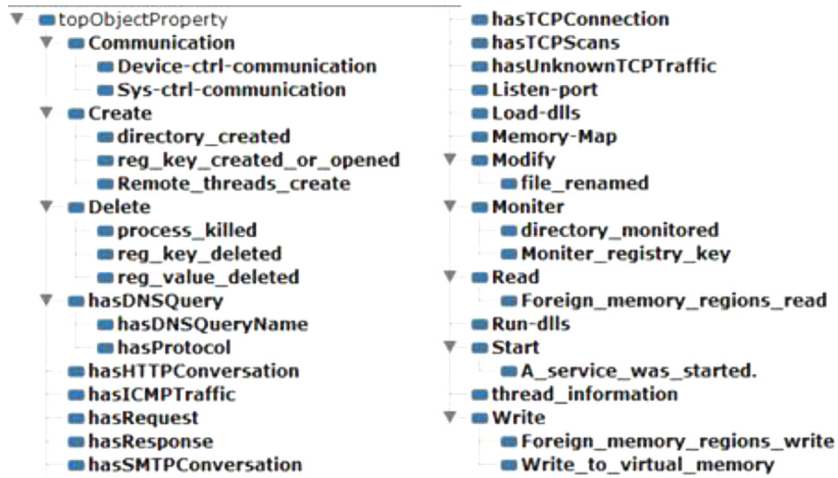


Fig. 4. Object properties of malware ontology.

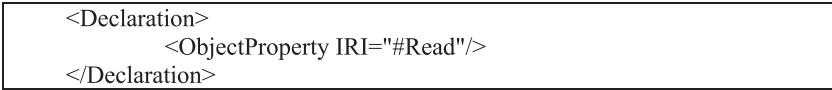


Fig. 5. Using OWL to define object property "Read".

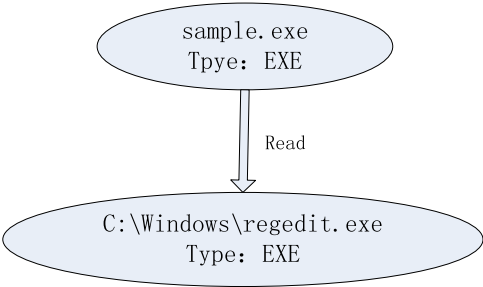


Fig. 6. Semantic description for object property "Read".

between malware class and computer system components class. In the following sections, we describe how to design the core classes in detail.

3.2. Design of malware class and system component class

As shown in Fig. 1, malware class and system component class are two core classes in malware ontology, which represent the basic concepts in malware domain. The malware class defines the concepts about malware categories, malware families and malware individuals. The system component class defines the concepts about the categories and individuals of system components. In the study all malware comes from Windows platform, so the ontology model is designed only for Windows malware.

Malware can be divided into different categories. Hansman and Hunt (2005) divided malware into seven classes, which are virus, worm, Trojan, back door, rogue software, time bomb and logic bomb. In the paper (Simmonds et al., 2004), malware is classified into five categories, which are virus, Trojan, worm, back door and rootkit. In our work we choose the classification mechanism in Simmonds et al. (2004) and classify malware into five categories. Each malware category contains different malware sub-categories, and each sub-category contains different malware families. Therefore, the hierarchy of malware classes has four levels, malware (top class), malware category, malware sub-category (optional) and malware families. Fig. 2 shows the structure of malware classes.

In Fig. 2 each class represents a specific concept within the model. A class can be more general (upper class) or more specific (subclass), e.g. Net Worm is a sub-category of worm, Xpaj and Agobot are the families of Back door and Virus categories, respectively (sub-categories of Back Door and Virus are ignored). An ontology always has a most general class, by convention, this class is called Thing. In the prototype model we create about 200 malware families. Due to the limitation of malware samples, we only create behavior descriptions for eight families which are shown in Fig. 2. In Section 3.5 we discuss how to describe the behaviors of a family and how to extract family behaviors.

In addition to malware classes, we also extract system component classes of malware domain. To achieve its purpose, malware needs to attack system components, such as deleting files, hiding processes, changing registry table, and IP routing table. System component classes consist of the targets malware will attack. Usually the hierarchy of system component classes has three levels, system component class (top class), system component category, and system component sub-category. If necessary, the sub-category can be further expanded. In the prototype model we define five system component categories: file class, network class, process class, registry class, and common component class (see Fig. 3). Fig. 3 shows all system component classes (about 13 sub-categories) in the ontology model.

File class defines the file types that malware will attack. Similarly, we can define the sub-categories of file class. In our work, the file class includes three sub-categories: directory, dynamic link library (dll) and execution file (exe). The registry class defines the registry information of malware attack, which contains one sub-categories, registry key. A registry key has two values: the key name and the key value. The process class defines the information of the process attacked by malware, which consists of two sub-categories: the affected process and the executable process. The affected process sub-category defines process information infected by malware, and the executable process sub-category defines the process information executed in memory. Network class contains network-related information about malware attacks, including sub-categories related to network activities. Currently we have defined the following sub-categories: DNS Query, HTTP Conversation,

IP, Port, Protocol and TCP Connection. Due to the complexity of network activity, the network class is more complex than other classes, and each sub-category can define its own sub-classes. For example, DNS Query contains the following sub-classes, Query Name, Query Type, Protocol Type and so on. In an ontology system, a subclass is not allowed to belong to different classes. To solve this issue, common component classes are designed to define shared classes for different system component classes. For example, the notify-filter sub-category can monitor the changes on file system and network, so it can be shared by file class and network class.

Once the core classes have been determined, we can define the individuals of each class. For example, Allapple is a family of the sub-category Net-worm (see Fig. 2). We can further insert leaf nodes under Allapple and these leaf nodes represent samples of a family. These sample nodes are called individuals. Malware family and malware individual are important concepts in malware domain. As we know, there are a huge amount of malware samples, and each sample has different behaviors. So the toughest problem in designing malware ontology is how to define malware individuals and families. In our work an individual is represented by its feature behaviors, and a malware family is defined as the common behaviors of samples in the family. In this section we only give the class structure of malware. We will discuss how to represent malware individuals and families in Sections 3.4 and 3.5.

The system component classes can be further expanded in the same way, for example, we can expand the class EXE, and add the specified file samples under it. These file samples are the individuals of EXE class. An individual is a concrete instance of a concept class. In malware detection field an individual can be a malware sample, a certain file, a process and so on.

3.3. Design of object property (behavior class)

Ontology is a formal representation of a set of concepts and their relationships in a particular domain. In this section, we define object properties that are used to represent relationships between concepts. From Fig. 1 we can see the behavior class contains the activities that malware performs on system components, and these activities reflect the relationship between malware individuals and system components. Therefore, object properties are represented as malware activities.

There are many different types of malware activities. In our study, malware behaviors are extracted from the behavior report generated by Anubis sandbox. In the prototype model, we only consider thirteen types of activities, including thirty eight activities. For example, a program can create a directory, create a registry, or create a remote thread, and these activities have the same type "Create". Based on the malware behaviors, we can design the corresponding object properties, as shown in Fig. 4. We have designed 38 object properties, belonging to thirteen property types.

From Fig. 4 we can see that the object properties mainly include creation, modification, reading, deleting, loading, running, monitoring, memory mapping and other attributes. Among these object properties, network properties are a bit complicated and contain many specific network activities, including listening port, DNS query, HTTP conversation, TCP connection, and more. Each object property has its own domain and range. Domain indicates which class an object property belongs to, and range indicates the value range of an object property. For example, the domain of object property Modify belongs to malware classes, and the range of Modify belongs to system component classes.

In malware ontology, object properties are defined according to malware behaviors. Here, we give two samples to discuss how to define object properties according to the behavior description. Table 1 shows a simple behavior description generated by Anu-

Table 1

Behavior report for file reading.

Malware name:	sample.exe
Read	C:\Windows\regedit.exe

Table 2

Behavior report for DNS query.

Malware name: NVC32.EXE		
DNS Queries	Query_Name:	rx7.teensmutbox.com
	Query_Type:	DNS_TYPE_A
	Successful:	NO
	Protocol:	UDP

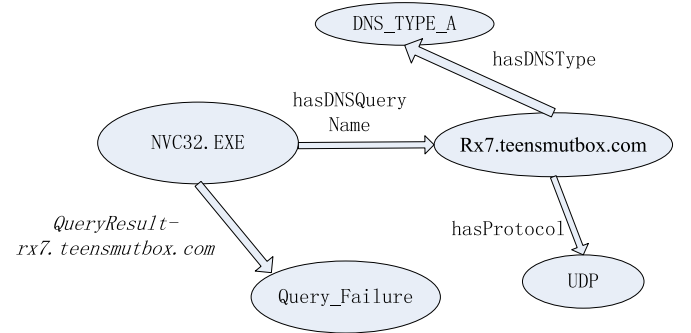


Fig. 7. Semantic descriptions for DNS Query behavior.

bis sandbox, in which malware sample.exe reads file regedit.exe. Corresponding to this behavior, we declare an object property "Read" to express file reading. Fig. 5 gives the definition of the object property "Read" described using Web Ontology Language (OWL). Fig. 6 gives the semantic link of object property "Read". In Section 3.4 we discuss how to use triples to describe the semantic of a behavior.

Table 2 shows a DNS query behavior of malware "NVC32.EXE". For this behavior, we need to pay attention to the fact that, on the one hand, different malware may use the same query to search a DNS server and return different query results. On the other hand, a malware may use different queries to search one DNS server. To distinguish different queries and different query results, we design annotation properties, and use multiple object properties to describe this behavior.

According to Table 2, we divide a DNS query behavior into different sub-behaviors (the links in Fig. 7), and we define an object property for each sub-behaviors. For example, the sub-behavior "QueryResult" represents obtaining the result of a DNS Querys. We design an annotation property "QueryResult" to represent this behavior, and use "rx7.teensmutbox.com" as an auxiliary description to distinguish different query results. In the same way we can design other object properties.

The object properties corresponding the behavior "DNS query" are defined in Fig. 8. From this sample, we can see when designing the object properties for complex behaviors, complex behaviors should first be divided into different sub-behaviors, and then the corresponding object properties are defined for each sub-behaviors.

3.4. Ontology description of malware individual

In the ontology model a malware sample is an individual belonging to a certain malware family. To obtain malware knowledge, we first need to represent the knowledge of malware individuals. Usually security experts determine whether a program is malicious or not by observing its behaviors. Therefore, malware behaviors are

<pre> <Declaration> <ObjectProperty IRI="#hasDNSQueryName"/> </Declaration> <Declaration> <AnnotationProperty IRI="#hasDNSType"/> </Declaration> <Declaration> <AnnotationProperty IRI="#QueryResult-rx7.teensmutbox.com"/> </Declaration> <Declaration> <AnnotationProperty IRI="#hasProtocol"/> </Declaration> </pre>

Fig. 8. Defining object properties for DNS Query behavior.

<pre> <Declaration> <NamedIndividual IRI="#sample.exe"/> </Declaration> <ClassAssertion> <Class IRI="#EXE"/> <NamedIndividual IRI="# sample.exe "/> </ClassAssertion> <ObjectPropertyAssertion> <ObjectProperty IRI="#Read"/> <NamedIndividual IRI="#sample.exe"/> <NamedIndividual IRI=" C:/Windows/regedit.exe "/> </ObjectPropertyAssertion> </pre>
--

Fig. 9. Description of the individual in Fig. 4 using OWL.

valuable information for malware analysis. In the ontology model, we use malware behaviors to represent the knowledge of malware individuals.

How do we describe malware behavior? When using natural language to describe malware behavior, the sentence pattern is "Malware A attacks system component B". Therefore, a malware behavior can be represented as a triple <subject, predicate, object>, which includes two concepts and the relationship between them. The subject and object in the triple are the concepts from a knowledge domain, which correspond to concept classes in ontology. Usually all classes or individuals can be the subject of a triple. A behavior acts on system components, so the object is usually represented as a system component class. The predicate describes the relation between the subject and the object, and we use the object property to represent it. In the ontology model we use OWL to formally describe a triple. The simple behavior can be expressed directly as a single triple. For the complex behaviors, such as network behavior, we need to use several triples to describe it.

Table 1 shows the behavior of a sample named sample.exe, in which sample.exe reads file regedit.exe. This behavior can be described as a single triple, in which the subject is sample.exe (an individual of subclass EXE), the predicate is Read (an object property), and the object is regedit.exe (an individual of subclass EXE). We use object property "Read" to describe this behavior. The domain of "Read" is the subject sample.exe and the range of "Read" is the object regedit.exe. If the sample only has this behavior, the individual sample.exe represented in OWL is shown as Fig. 9.

Table 2 shows the behaviors of the sample NVC32.exe. In Section 3.3 we define the object properties for describing a DNS query behavior. We can use these object properties to define malware "NVC32.EXE". We use multiple triples to describe this behavior (see Fig. 7). For example, in Fig. 7 "QueryResult-rx7.teensmutbox.com" is an annotation property which stands for the query result of the DNS Query. The subject of the

annotation property is "NVC32.EXE", which is used to identify the individual sending the query request. From the two triples <rx7.teensmutbox.com, hasDNSType, DNS_TYPE_A>, and <NVC32.exe, hasDNSQueryName, rx7.teensmutbox.com>, we can see that an object of one triple could be a subject of another triple.

Fig. 10 gives the definition of individual "NVC32.EXE" in OWL. In Fig. 10, UDP is an individual of subclass Protocol, DNS_TYPE_A is an individual of Query_Type, NO is an individual of Query_Successful. Because the DNS Query is not successful (return NO), there is no query result. For the convenience of understanding, we use Query_Failure to represent it, in fact it can be ignored. From this sample, we can see multiple object properties can be combined to describe a complex behavior. Firstly we need to separate a complex behavior into different triples, and then define the object, subject and property of each triple. In Fig. 10 the description < Class IRI="#Agobot"/> shows the family of NVC32.EXE is "Agobot".

3.5. Ontology description of malware family and reasoning

In Section 3.2 we define the class structure of malware. Just like malware individuals, we need to characterize behavior features of each malware category. Among these categories the definition of malware family occupies an important position. Family knowledge can help researchers analyze the common characteristics of family members and discover the relationship between different families. In the class structure of malware, each malware family is a superclass of its individuals, and each individual is defined by its behaviors. In order to obtain the knowledge of each family, we extract the common behaviors of the samples from the same family and use these common behaviors to define a malware family.

In our work, Apriori algorithm (Agrawal and Srikant, 1994) is applied to mine the common behavior of a malware family. Apriori is an algorithm for frequent item set mining and association rule

<pre> <Declaration> <NamedIndividual IRI="# NVC32.EXE "/> </Declaration> <ClassAssertion> <Class IRI="#Agobot"/> <NamedIndividual IRI="# NVC32.EXE "/> </ClassAssertion> <ObjectPropertyAssertion> <ObjectProperty IRI="#hasDNSQueryName"/> <NamedIndividual IRI="# NVC32.EXE "/> <NamedIndividual IRI="#rx7.teensmutbox.com"/> </ObjectPropertyAssertion> <AnnotationAssertion> <AnnotationProperty IRI="#hasDNSType"/> <IRI>#rx7.teensmutbox.com</IRI> <IRI>#DNS_TYPE_A</IRI> </AnnotationAssertion> </pre>
<pre> <AnnotationAssertion> <AnnotationProperty IRI="#QueryResult-rx7.teensmutbox.com"/> <IRI># NVC32.EXE </IRI> <IRI># QueryFailure </IRI> </AnnotationAssertion> <AnnotationAssertion> <AnnotationProperty IRI="#hasProtocol"/> <IRI>#rx7.teensmutbox.com</IRI> <IRI>#UDP</IRI> </AnnotationAssertion> </pre>

Fig. 10. Description of the individual in Fig. 3 using OWL.

learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. A frequent item set (denoted as L_k , where k is the size of the itemset) is an item set whose support is greater than some user-specified minimum support. The support of an itemset denoted as $\text{sup}(\{i_1, \dots, i_{k-1}\})$ is the frequency of the itemset appears in a transactional database. An itemset $\{i_1, \dots, i_{k-1}, i_k\}$ can be written as a rule, for example, $\{i_1, \dots, i_{k-1}\} \rightarrow i_k (\%s, \%c)$, where $s\%$ (the support of the rule) is the support of the items i_1, \dots, i_k occur together, and $c\%$ (the confidence of the rule) is calculated as $\text{sup}(\{i_1, \dots, i_k\}) / \text{sup}(\{i_1, \dots, i_{k-1}\})$. Both $s\%$ and $c\%$ are obtained by calculating the frequency of the respective itemsets in a given transactional database DB . Interesting rules are the rules with $s\%$ and $c\%$ greater than or equal to the user-specified minimum support (denoted as ms) and the minimum confidence (denoted as mc), respectively.

The malware individuals belonging to the same family are grouped together, and each group can be seen as a transactional database. Before Apriori algorithm mining the database, each individual in the database needs to be represented as an item set. Some examples of malware item sets are shown in Table 3. In Table 3, the individuals come from family A, and each row (also called a transaction) gives all behaviors of a sample. We use one or more items to represent a behavior, each of which corresponds to a triple. For convenience, we use the property and object of a triple to represent an item, and the two parts are connected by the symbol '#'. For simple behavior, we can directly convert a triple into an item. For a complex behavior, we divide it into different triples, and convert each triple into an item. For example, the behavior DNS Query is represented as four items which are shown in the third row of Table 3. After transforming individuals into item

Table 3

A transaction database for malware family A.

Samples	Item set
Malware 1	Memory-Map# WindowsShell.Mainfest
	Read#HKLM/SYSTEM/CurrControlSet/Serv/Winsock/Para.:Transports Load-dll# SHLWAPI.dll
	...
Malware 2	hasDNSType#DNS_TYPE_A
	hasDNSQueryName# rx7.teensmutbox.com
	hasProtocol#UDP
	queryResult- rx7.teensmutbox.com#Query_Failure
	...
...	...

sets, Apriori algorithm can be applied to mine the frequent items. The frequent items in a database can be regard as the common behaviors of a malware family. Finally, the common behaviors are represented using ontology language, which are used to define the family class.

One key problem for Apriori algorithm is determining the length of an item set. The Apriori property states that all nonempty subsets of a frequent item set are also frequent item sets. So, if $\{A, B\}$ is a frequent itemset then subsets $\{A\}$ and $\{B\}$ are also frequent item sets. For malware detection, if a malware sample matches with a frequent itemset $L_k = \{i_1, i_2, \dots, i_k\}$, it will also matches with an item $L_{k-1} = \{j_1, j_2, \dots, j_{k-1}\}$, and $\{j_1, j_2, \dots, j_{k-1}\}$ is a subset of $\{i_1, i_2, \dots, i_k\}$. So if we use a subset of L_k (denoted as L_m , and $m < k$) to define the features of a kind of malware, we found that as m decreases, more and more test samples can match with L_m , which results an increase in false alarm rate. To determine the length of the frequency itemset for malware detection, we prepare

```

<Declaration>
  <Class IRI="# family A "/>
</Declaration>
<SubClassOf>
  <Class IRI="#Parent Class of family A "/>
</SubClassOf>
<ObjectPropertyAssertion>
  <ObjectProperty IRI="# Memory-Map "/>
  <NamedIndividual IRI="# family A "/>
  <NamedIndividual IRI="# WindowsShell.Manifest "/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
  <ObjectProperty IRI="# Load-dll "/>
  <NamedIndividual IRI="# family A "/>
  <NamedIndividual IRI="# SHLWAPI.dll "/>
</ObjectPropertyAssertion>

```

Fig. 11. Description of family A using OWL.


```

<Declaration>
  <Class IRI="# family A "/>
</Declaration>
<SubClassOf>
  <Class IRI="#Parent Class of family A "/>
</SubClassOf>
<EquivalentClasses>
  <Class IRI="# family A "/>
  <ObjectIntersectionOf>
    <ObjectSomeValuesFrom>
      <ObjectProperty IRI="#Load-dlls"/>
      <Class IRI="# SHLWAPI.dll "/>
    </ObjectSomeValuesFrom>
  </ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
  <Class IRI="# family A "/>
  <ObjectIntersectionOf>
    <ObjectSomeValuesFrom>
      <ObjectProperty IRI="# Memory-Map "/>
      <Class IRI="# WindowsShell.Manifest "/>
    </ObjectSomeValuesFrom>
  </ObjectIntersectionOf>
</EquivalentClasses>


```

Fig. 12. Defining EquivalentClasses of family A using OWL.

Description: Agobot

Equivalent To 

- (Load-dlls some pre1:WS2HELP.dll)
 - and (Load-dlls some pre1:WS2_32.dll)
 - and (Load-dlls some pre1:kernel32.dll)
 - and (Load-dlls some pre1:ntdll.dll)
 - and (Load-dlls some pre2:comctl32.dll)
 - and (Memory-Map some Windows Shell.Manifest)
 - and (Read some HKLM/System/CurrentControlSet/Control/Terminal_Server:TUserEnabled)
 - and (Read some HKLM/System/Setup:SystemSetupInProgress)
- (Load-dlls some pre1:WS2_32.dll)
 - and (Load-dlls some pre1:kernel32.dll)
 - and (Load-dlls some pre1:ntdll.dll)
 - and (Load-dlls some pre2:comctl32.dll)
 - and (Memory-Map some Windows Shell.Manifest)
 - and (Read some HKLM/Software/Policies/Microsoft/Windows/Safer/CodeIdentifiers:TransparentEnabled)
 - and (Read some HKLM/System/CurrentControlSet/Control/Terminal_Server:TUserEnabled)
 - and (Read some HKLM/System/Setup:SystemSetupInProgress)

SubClass Of 

- Back_Door

Fig. 13. Defining the EquivalentClasses for family Agobot using Protégé.

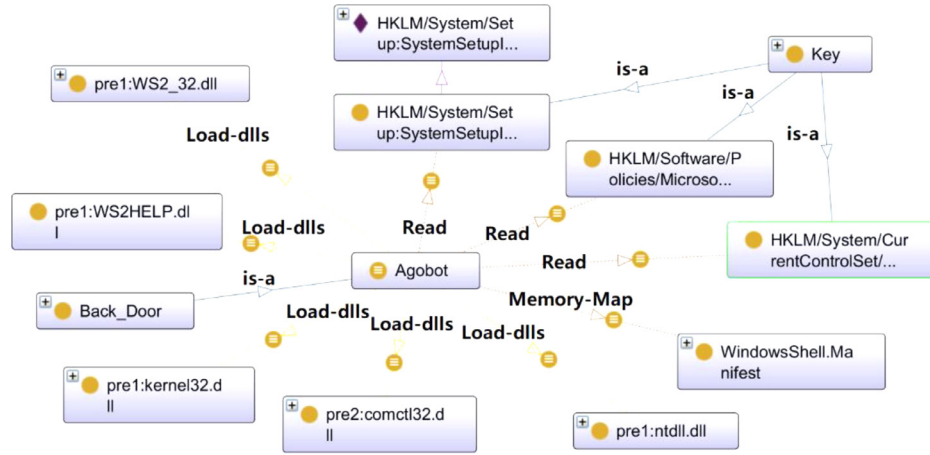


Fig. 14. Part of semantic links of family Agobot.

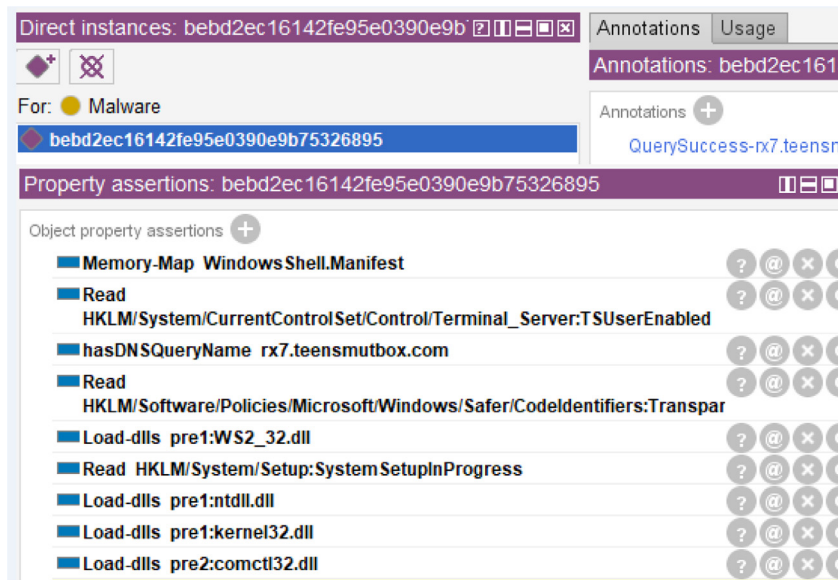


Fig. 15. Defining an individual using Protégé.

a validation set containing malicious and benign samples. The validation set is used to evaluate the performance of L_k . Usually we control the false alarm rate below a threshold, and then select L_k with the highest detection rate. The common behaviors of a malware family are defined as the items in the frequent item set L_k . Apriori algorithm is a time consuming algorithm. To improve the efficiency of the algorithm, we need to choose some behaviors for data mining. In reality we can set a threshold, and only choose the behaviors that appears in the number of samples bigger than the threshold.

The frequent items in L_k are the common behaviors of a family, which can be used to represent a family class in ontology model. In fact the properties and objects in the items of L_k are already defined in malware ontology. Therefore, it is easy to represent the frequent items (common behaviors of a malware family) using ontology language. For example, the frequent items with length one for family A is as follows:

```
Memory-Map# WindowsShell.Manifest
Load-dll# SHLWAPI.dll
```

then, the family A represented in ontology language is shown as Fig. 11.

The purpose of mining common behaviors is to classify malware families. The ontology language OWL provides the EquivalentClass axiom that allows one to state that several class expressions are equivalent to each other. We define the EquivalentClasses of a family and use the EquivalentClasses to infer the family of a malware sample. For example, the EquivalentClasses of family A described in OWL is shown as Fig. 12. In Fig. 12 each frequent item in L_k is defined as an EquivalentClass axiom. If the behaviors of an individual match with an EquivalentClass, then the individual is classified as family A. In addition, we can also use Protégé to check the consistency of malware knowledge. For example, we define two disjoint classes, family A and family B, if an individual can be classified into both families, then the reasoner can check such inconsistency.

In our study we use the ontology editor Protégé (Protégé, 2015) to build malware ontology. Protégé provides a graphic user interface to define ontologies. We can easily build class, object properties, individuals by editing different entries provided by Protégé. Protégé includes deductive classifiers to infer new information based on the analysis of ontology. For an unknown sample, we firstly run it in the Anubis sandbox, and obtain its behavior report, and then we create an individual according to the behavior report of the sample. Protégé provides an interface to cre-

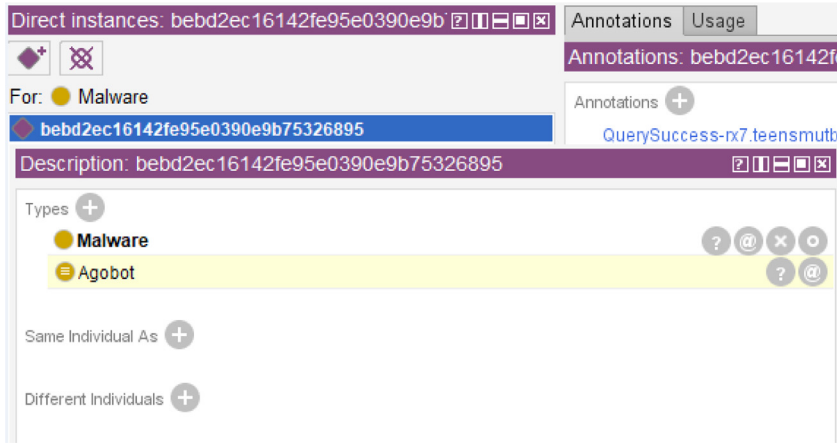


Fig. 16. Inferring the family of an individual.

Table 4

A transaction database for malware family B.

No. of frequent itemsets	Frequent itemsets
1	Load-dlls#C:/WINDOWS/system32/kernel32.dll Load-dlls#C:/WINDOWS/system32/ntdll.dll Memory-Map#C:/WINDOWS/WindowsShell.Manifest
2	Load-dlls#C:/WINDOWS/system32/ntdll.dll Memory-Map#C:/WINDOWS/WindowsShell.Manifest hasDNSType#DNS_TYPE_A

Table 5

classification rules for malware family B.

Rule no	Item set
1	r1: [(?x Load-dlls C:/WINDOWS/system32/kernel32.dll) (?x Load-dlls C:/WINDOWS/system32/ntdll.dll) (?x Memory-Map C:/WINDOWS/WindowsShell.Manifest) -> (?x rdf:type B)]
2	r2: [(?x Load-dlls C:/WINDOWS/system32/ntdll.dll) (?x Memory-Map C:/WINDOWS/WindowsShell.Manifest) (?x hasDNSType DNS_TYPE_A) -> (?x rdf:type B)]

ate an individual, and we only need to add description of behaviors. Thus we can use the reasoner provided by Protégé to deduce the relationships between classes or between class and individual. For example, In Fig. 13 we use Protégé to define the EquivalentClasses of the family Agobot. Common behaviors of the family are shown in the EquivalentClasses. Fig. 14 shows part of semantic links of family Agobot. In Fig. 15 an unknown individual named “bebd2ec16142fe95e0390e9b75326895” is created using Protégé, and the object properties are used to describe its behaviors. After creating the individual, we start the reasoner (a menu button in the Protégé Tool). The reasoner can match the behavior of the individual with that of all malware families, and infer the family the individual belongs to. In Fig. 16 the reasoner infers the individual’s family is “Agobot”.

The common behavior of a malware family can also be represented as rule-based knowledge, and we can use a rule based reasoner to recognize the family of an unknown malware. Table 4 shows two frequent itemsets of malware family B, and each itemset includes three behaviors. Table 5 shows the family rules corresponding to each frequent itemset. One rule is created for each frequency itemset, each item (a behavior) of the itemset becomes a rule antecedent, and the body part of a rule (left part of “->”) is

Table 6

Description of malware families.

Family name	Variants	Samples
Allapple	4	280
Mydoom	3	270
Mytob	51	290
Bagle	23	275
Netsky	40	285
Agobot	32	309
Apaj	2	260
Zbot	4	260

the conjunction of rule antecedents. The family category becomes the rule consequent (the right part of “->”). The symbol “?x” is a rule variable, which is an unknown malware instance. In our work, we design a rule generator, which automatically translate each frequent itemset into a rule described in ontology language. The malware ontology in Protégé is exported to a compatible structure with Jena, then we use Jena reasoner to infer the family of an unknown malware.

3.6. Experiments for family classification

In Section 3.4 we use data mining method to extract the common behavior of a malware family. In this section we evaluate the accuracy of family classification using this method. For the convenience of experiment comparison, we choose the same dataset as that used in the paper (Ding et al., 2018).

3.6.1. Description of experimental datasets

In the experiments part of the experimental data comes from the website (Malware, 2019). To obtain more samples of different families, we use the anti-virus tool Kapasky to test more malware samples, and use the malware name assigned by Kapasky to decide if samples belong to the same family. A malware name assigned

Table 7
Data partitions for the experiments.

Dataset	Training set (for extracting family behavior)	Testing set		Validation set	
		Benign	Malware	Benign	Malware
Allaple	60	300	150	150	70
Mydoom	60	300	140	150	70
Mytob	60	300	160	150	70
Bagle	60	300	145	150	70
Netsky	60	300	155	150	70
Agobot	60	300	179	150	70
Apaj	60	300	130	150	70
Zbot	60	300	130	150	70

Table 8
Detection results of DM and CDG methods.

Family name	DM method							CDG method			
	TPR (%)	FPR (%)	ACC (%)	CLTime (ms)	TrTime (s)	Com. Beh.	Beh. No.	TPR (%)	FPR (%)	ACC (%)	CLTime (ms)
Allaple	100	0.0	100	7.0	0.4	89 × 1	131	100	0.0	100	3.4
Mydoom	100	1.6	98.9	6.8	50.1	22 × 1	160	95.5	0.0	98.6	5.8
Mytob	83.0	1.5	93.1	8.6	0.4	10 × 66	263	86.4	0.0	95.3	8.1
Bagle	94.0	1.6	97.0	8.9	0.03	26 × 1	112	92.7	0.0	97.6	8.1
Netsky	87.0	6.6	91.2	7.9	0.08	8 × 45	121	75.4	0.0	91.7	5.7
Agobot	93.8	3.5	95.5	6.6	2.0	8 × 45	152	87.6	0.0	95.4	6.3
Xpaj	88.5	2.0	95.1	7.8	0.3	10 × 66	159	86.2	0.0	95.8	6.6
Zbot	95.4	3.0	96.5	7.4	0.4	12 × 35	123	91.5	0.0	97.4	6.5
Average	92.7	2.5	96.0	7.6	6.7			89.6	0.0	96.5	6.3

by Kapasky includes different parts. For example, in the malware name Backdoor.Win32.Agobot.nq, the last suffix of the name is the variant name, and the suffix before the variant name is the family name. So we can collect malware samples from the same family according to the family name. We collected the malware samples coming from eight families, which are shown in Table 6. We also collected benign samples which are randomly chosen from several desktop workstations that run Windows 7.

The data partitions for the experiments are shown in Table 7. The training set for each malware family is used to mine family behavior. The testing sets are used to test the final performance of the data mining method. In Table 6, the benign testing sets for all of the malware families are the same, and the malware samples in the training set and the testing set come from the same family, but they are totally different. To decide the length of frequent itemset, we prepare a validation set for each family (another dataset different from the training and testing dataset), which include 70 malicious samples from the same family, and 150 benign samples.

In order to extract the common behavior of the family, we need to collect enough samples from the same family. In fact, it is difficult to determine the number of samples used to extract common behaviors. Usually the training dataset should include the samples from each variants in the family. We plan to extract the common behaviors for 200 families. However, we do not collect enough samples for most families (only four to five samples are collected for most families). Therefore, in the ontology model we only extract the common behaviors for eight families.

In the experiment, we repeat each experiment 12 times and calculate the average accuracy to evaluate the classification performance. For each repeated experiment, the data partitions are the same, which is shown in Table 6. In each repeated experiment, 60 training samples are randomly selected, and the remaining samples are for testing. The behaviors appearing in more than 20% training samples are selected to mine family behaviors. The experimental environment is the Windows 10 operating system, with Intel i7 8700K 3.7 GHz CPU and 8 GB of main memory.

3.6.2. Experimental results

We used the following measurements to evaluate the proposed method: the true positive rate (TPR), the false positive rate (FPR), and the accuracy (ACC).

We define TP as the number of malicious samples classified as malware and TN as the number of benign samples classified as benign, while FP is the number of benign samples classified as malware, and FN is the number of malicious samples classified as benign (if a sample does not match with any EquivalentClass or behavior rule of a family, it is classified as benign). Thus, TPR, FNR, and FPR are defined as follows:

$$TPR = TP / (TP + FN) \quad (1)$$

$$FPR = FP / (TN + FP) \quad (2)$$

$$ACC = (TP + TN) / (TP + FN + TN + FP) \quad (3)$$

We compare the performance of the data mining method (DM) with the graph based method (CDG) in the paper (Ding et al., 2018). Both methods use the same dataset for training and testing (see Table 7). The experimental results of both methods are shown in Table 8. In Table 8 the column “CLTime” gives the family classification time, the column “TrTime” gives the time for extracting family behaviors from the training samples, the column “Beh. NO.” gives the total number of the behaviors used for mining family behaviors, and the column “Com. Beh.” gives the behavior number of a family (cxb, b is the number of rules or EquivalentClasses, and c is the number of behaviors in a rule or an EquivalentClass). In the experiments we describe common behaviors as rules, and use the rule based engine Jena to infer the families of the testing samples.

From Table 8 we can see the overall performance (ACC) of these two methods is very similar. CDG method uses API graph to represent malware behaviors. Compared with API based behaviors, the system behaviors used by DM are high level and

coarse-grained behaviors. For the same behavior occurred in benign and malicious samples, the DM method uses the same representation to describe it. This results a relatively high FPR. For example, there are two samples, one sample from the family "Allaple" and the other from benign samples. They all reads window registry to retrieve user's profile information. In DM method the two samples have the same behavioral representation "read#HKEY_LOCAL_MACHINE\...\ProfileList". If the common behaviors of the family "Allaple" include this behavior, it is very likely that the benign sample is misclassified. The CDG method uses API dependency graph to represent behavior. The above behavior can be represented as an API sequence "RegOpenKey(...) RegQueryValueEx(...)". Usually the benign sample and the malicious sample process the registry information in different ways. This results the APIs related with the above API sequence in the malicious sample are different from that in the benign sample. Therefore, the CDG method will use two different API dependency graphs to represent the two samples.

From the experimental results we can see that Apriori algorithm can discover the common behaviors of malware family, and the common behaviors mined can effectively recognize the family of malware samples.

4. Conclusions

In this paper we introduce ontology techniques into the malware domain, and propose the method for constructing malware knowledge base. Our contributions are as follows:

We design the concept classes and object properties of malware, and propose the method for representing malware behaviors.

We propose the methods for constructing the knowledge of malware individuals and malware families. The knowledge of malware individuals is represented as their behaviors, and the knowledge of malware family is described as the common behaviors of family members.

We design the method for extracting the common behaviors of individuals belonging to the same family. The Apriori algorithm is used to extract the common behaviors of a malware family. The experimental results show that the common behaviors mined using Apriori algorithm can effectively recognize the family of malware samples.

The malware ontology stores detailed behavior knowledge about malware individuals and families. As a knowledge tool, it provides an effective way for researchers to analyze malware samples, understand infection procedures of malware, and measure the potential damage extent of malware. For example, computer virus spread by copying themselves. Through the analysis of virus behaviors (file read and file write), and the modified files, we can understand the process of virus infection. If a malware establishes a large amount of TCP or UDP connections, it can be evaluated to be more destructive. Just like users can use different calculation method to obtain the statistical information from a traditional database. The malware ontology stores the knowledge, which is machine readable and understandable, so researchers can develop their own analyzing engine to obtain the knowledge they want. However, in the ontology model malware behavior is described as coarse-grained behaviors (system behaviors). Compared with fine-grained behaviors (API behaviors), system behaviors cannot provide the accuracy information about the infection process of virus and the damage extent of malware.

In present only a limited number of concept classes are created in the prototype model. We still need to further improve the model. Our future work includes further extending the concept classes, studying the classification architecture of malware, and designing a high efficient algorithm to extract common behaviors of a malware family.

Declaration of Competing Interest

No conflict of interest.

Acknowledgements

This work was partially supported by Scientific Research Foundation in Shenzhen (Grant No. JCYJ20180306172156841, JCYJ20180507183608379), Guangdong Natural Science Foundation (Grant No. 2016A030313664), the National Natural Science Foundation of China (Grant No. 61872107) and the National Key R&D Program of China (Grant No. 2018YFB1003800, 2018YFB1003805).

References

- Abdoli, F., Meibody, N., Bazoubandi, R., 2010. An attacks ontology for computer and networks attack. In: *Innovations and Advances in Computer Sciences and Engineering*. Springer, Netherlands, pp. 473–476.
- Agrawal, R., Srikant, R., September 1994. Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, Santiago, Chile*, pp. 487–499.
- Barry, S., Christopher, W., 2001. Ontology-towards a new synthesis. In: *Proceedings of the International Conference on Formal Ontology in Information Systems*, 2001, pp. 3–9.
- Chiang, H.S., Tsar, W.J., 2010. In: Elmagarmid, A.K., Agrawal, D. (Eds.), *Mobile Malware Behavioral Analysis and Preventive Strategy Using Ontology*. SocialCom, pp. 1080–1085.
- Ding, Y., Xia, X., Chen, S., Li, Y., March 2018. A malware detection method based on family behavior graph. *Comput. Secur.* 73, 73–86.
- Filiol, E., 2006. Malware pattern scanning schemes secure against blackbox analysis. *J. Comput. Virol.* 2 (1), 35–50.
- Hansman, S., Hunt, R., 2005. A taxonomy of network and computer attacks. *Comput. Secur.* 24 (1), 31–43.
- Huang, H.D., Lee, C.S., Wang, M.H., et al., 2014. IT2FS-based ontology with soft-computing mechanism for malware behavior analysis. *Soft Comput.* 18 (2), 267–284.
- Jasiul, B., Sliwa, J., Gleba, K., Szpyrka, M., 2014. Identification of malware activities with rules. In: Ganzha, M., Maciaszek, L.A., Paprzycki, M. (Eds.), *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, Warsaw, Poland, September 7–10, pp. 101–110.
- Malware L. (2019). Malicious code samples available from: <http://malware.lu> (Accessed 20 May 2019)
- Martinez, C.A., Echeverri, G.I., Sans, A.G.C., 2010. Malware detection based on cloud computing integrating intrusion ontology representation. In: 2010 IEEE Latin-American Conference on Communications (LATINCOM), pp. 1–6.
- Michael, I., et al., 2015. Developing an ontology for cyber security knowledge graphs. In: *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, pp. 1–4.
- More, S., Matthews, M., Joshi, A., 2012. A knowledge-based approach to intrusion detection modeling. In: *IEEE Symposium on Security and Privacy Workshops (SPW)*, pp. 75–81.
- Mundie, D.A., McIntire, D.M., 2013. An ontology for malware analysis. In: 2013 International Conference on Availability, Reliability and Security, Regensburg, Germany, September 2–6, pp. 556–558.
- Navarro, L.C., Navarro, A.K.W., Grégio, A., Rocha, A., Dahab, R., 2018. Leveraging ontologies and machine-learning techniques for malware analysis into android permissions ecosystems. *Comput. Secur.* 78, 429–453.
- Obrst, L., Chase, P., Markeloff, R., 2012. Developing an ontology of the cyber security domain. In: da Costa, P.C.G., Laskey, K.B. (Eds.), *Proceedings of the Seventh International Conference on Semantic Technologies for Intelligence, Defense, and Security*. CEUR Workshop Proceedings, 966, pp. 49–56.
- Protege, 2015, <http://protege.stanford.edu/products.php#web-protege>.
- Shoaib, M., Farooq, M., 2015. Uspam – a user centric ontology driven spam detection system. In: 2015 48th Hawaii International Conference on System Sciences (HICSS), pp. 3661–3669.
- Simmonds, A., Sandilands, P., Van Ekert, L., 2004. An ontology for network security attacks. In: *Applied Computing*. Springer, Berlin Heidelberg, pp. 317–323.
- Tafazzoli, T., Adjani, S.H., 2008. Malware fuzzy ontology for semantic web. *Int. J. Comput. Sci. Netw. Secur.* 8, 153–161.
- Undercoffer, J., Joshi, A., Pinkston, J., 2003. Modeling computer attacks: an ontology for intrusion detection. In: *Recent Advances in Intrusion Detection*. Springer, Berlin Heidelberg, pp. 113–135.
- Undercoffer, J., Pinkston, J., Joshi, A., et al., 2004. A target-centric ontology for intrusion detection. In: 18th International Joint Conference on Artificial Intelligence, pp. 9–15.
- Wang, P., Chao, K.M., Lo, C.C., Wang, Y.S., 2015. Using ontologies to perform threat analysis and develop defensive strategies for mobile security. *Inf. Technol. Manag.* 1–25.
- Wang, W., Li, Y., Wang, X., Liu, J., Zhang, X., 2018. Detecting android malicious apps and categorizing benign apps with ensemble of classifiers. *Future Generation Computer Systems* 78, 987–994.
- Wang, W., Zhao, M., Gao, Z., Xu, G., Xian, H., Li, Y., Zhang, X., 2019. Constructing features for detecting android malicious applications: issues, taxonomy and directions. *IEEE Access*. <https://www.doi.org/10.1109/ACCESS.2019.2918139>.

Further Reading

Grégio, A., et al., 2016. An ontology of suspicious software behavior. *Appl. Ontol.* 11 (1), 29–49.

Salomon, D., 2006. *Foundations of Computer Security*. Springer, London.

Yuxin Ding received the Ph.D. degree in computer science from Institute of Software, Chinese Academy of Sciences, in 1999. He is currently an Associate Professor in the Department of Computer Science at the Harbin Institute of Technology Shenzhen Graduate School. His current research interests are primarily in computer security and machine learning.

Rui Wu received her B.S. degrees in Computer Sciences from the Jiangxi University in 2017. She is currently a master student in the Department of Computer Science at the Harbin institute of technology Shenzhen Graduate School. Her current research interests are in machine learning and computer security.

Xiao Zhang received his B.S. degrees in Computer Sciences from the Qingdao University in 2018. He is currently a master student in the Department of Computer Science at the Harbin institute of technology Shenzhen Graduate School. His current research interests are in machine learning and computer security.