Fan LJ, Wang YZ, Li JY *et al.* Privacy Petri net and privacy leak software. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 30(6): 1318–1343 Nov. 2015. DOI 10.1007/s11390-015-1601-7

Privacy Petri Net and Privacy Leak Software

Le-Jun Fan [1] (范乐君), *Member, IEEE*, Yuan-Zhuo Wang [2,*] (王元卓), *Senior Member, CCF, ACM, IEEE*
Jing-Yuan Li [2] (李静远), Xue-Qi Cheng [2] (程学旗), *Distinguished Member, CCF*, and
Chuang Lin [3] (林 闯), *Fellow, CCF, Senior Member, IEEE*

[1] *National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029, China*
[2] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China*
[3] *Department of Computer Science and Technology, Tsinghua University, Beijing 100083, China*

E-mail: fanlejun@cert.org.cn; {wangyuanzhuo, lijingyuan, cxq}@ict.ac.cn; chlin@tsinghua.edu.cn

Received March 27, 2014; revised June 2, 2015.

Abstract Private information leak behavior has been widely discovered in malware and suspicious applications. We refer to such software as privacy leak software (PLS). Nowadays, PLS has become a serious and challenging problem to cyber security. Previous methodologies are of two categories: one focuses on the outbound network traffic of the applications; the other dives into the inside information flow of the applications. We present an abstract model called Privacy Petri Net (PPN) which is more applicable to various applications and more intuitive and vivid to users. We apply our approach to both malware and suspicious applications in real world. The experimental result shows that our approach can effectively find categories, content, procedure, destination and severity of the private information leaks for the target software.

Keywords privacy Petri net, privacy leak software, privacy function, private information, malware analysis

1 Introduction

Internet and various kinds of software increase the relationship of people and make their life convenient, though lots of private information is collected and spread without proper protection. The private information can be used to invade and leak people's privacy by malware authors or illegal software providers for evil purposes. In fact, private information leak behavior has been widely discovered in a lot of malware and suspicious applications. We refer to such software as privacy leak software (PLS). Nowadays, PLS has become a serious and challenging problem to cyber security.

However, anti-virus software (AVS) companies do not value PLS properly, and do not have enough capability to handle PLS issues as well. PLS can easily avoid the detection of AVS and threaten the privacy security of the users. On one hand, some privacy leak behaviors are not identified as malicious by AVS; on the other hand, some PLS is not given enough information by AVS to let users understand their threat level. Therefore, it is necessary to find more effective methods for analyzing the behavior of PLS.

Recent research on analyzing software behaviors follows two kinds of road map: the black-box one focuses on the input data and the outbound network traffic of application, while the white-box one dives into the inside information flow of application. The approaches which focus on network traffic can rapidly find privacy data such as credit card number whose data format is well-defined[1-3]. But these approaches encounter the extreme complexity of packet obfuscating techniques such as encrypted connections, message reordering and traffic randomization. Other approaches which focus on insider information flow can analyze more accurate details about privacy leak behaviors[4-7]. These analyzing methods can be further divided into two categories: static analysis and dynamic analysis. The static

Regular Paper

This work is supported by the National Natural Science Foundation of China under Grant Nos. 61402124, 61402022, 61173008, 60933005, and 61572469, the National Key Technology Research and Development Program of China under Grant No. 2012BAH39B02, the 242 Projects of China under Grant No. 2011F45, and Beijing Nova Program under Grant No. Z121101002512063.

*Corresponding Author

analysis method finds the accurate data flow from binary executable file of a target application[8-11], but also confronts code obfuscating problems such as code morphing, packer and opaque constant[12-14]. Thus the dynamic method which gets the runtime data flow by tracing the execution of the target application is applied more widely[15-17] for application behavior analysis. However, dynamic analysis still has its disadvantages, such as the lack of ability to detect and cope with multiple paths[18] and dormant functionality[19].

To sum up, existing methods lack the ability for analyzing the behavior of PLS. The ideal methods should answer the following four questions which previous methods have not worked out about PLS at the same time.

- What is the content of the leaked information?
- Which kind of private information is leaked?
- How does an application leak the information?
- How serious is the leak behavior?

The goal of this paper is to propose a new model method to cover the above four questions and help the major software providers or end users to analyze and detect PLS. Early model methods focus on the formulation definition for software behavior modeling such as specification language or semantic template[20]. These methods can accurately describe certain software behavior, but the models are hard to build, understand or extend. This is the reason why graphic models such as control flow graph, behavior graph and hierarchical behavior graph are adopted[21-24]. Some high-level Petri net models such as stochastic Petri nets (SPN) and stochastic game nets (SGN) are also used to build quantitative analysis[25-26]. Nevertheless, there is still a lack of models which focus on privacy leak problem.

This paper presents an abstract model called Privacy Petri Net (PPN) to characterize the whole leaking procedure of PLS by more high-level description, making analysis results more applicable to most kinds of software and more meaningful to the users. PPN is a kind of high-level Petri net which focuses on PLS analysis and has three main features.

1) PPN provides a formal mathematical definition of syntax and semantics for privacy attribution and function calculation.

2) PPN has concise and powerful modeling primitives for graphical abstraction of privacy leak procedure.

3) PPN is modularized and can be used to build various hierarchical models.

Petri net is a useful tool for describing system status and calculating some system properties. In this paper, we consider a running program as an independent system. In this perspective, to find some specific behavior of this program can be considered as to find some specific status transformation of this system. Therefore, we apply Petri net to find the specific status transformation which stands for privacy leak behavior. More concretely, we depend on a useful property of Petri net called reachability to find the status transformation from a data access status to network transmission status.

We divide PLS into more accurate categories and trace the insider data flow. We also divide the execution of PLS into two phases: unauthorized data access and covert network transmission. We adopt PPN to model the two phases respectively and get the sub-model of different privacy leak behavior procedures.

We apply our approach on different kinds of real-world PLS. The experimental results show that our approach not only effectively detects private information leaks for the target software, but also finds the categories, content, procedure, destination and severity in detail.

The rest of this paper is organized as follows. Section 2 gives the overview of privacy leak behaviors. Section 3 discusses the definition of PPN. In Section 4, we model different leaks categories with PPN. Section 5 introduces our approach framework and algorithms. Section 6 shows the experiment on real-world PLS. Section 7 gives related work. Finally, Section 8 concludes the paper.

2 Privacy Leak

In this section, we give the details of the privacy leak by firstly giving the definition of privacy in this paper, and then discussing the details of privacy leak behaviors.

2.1 Privacy Leak Definition

It is hard to give an overall definition for privacy because of its diversity and ambiguity. In a broad sense, privacy is the independence of individual. In computer science, the privacy equals the data which contains private information and is called data privacy. Data privacy is an issue about how uniquely identifiable data related to a person or persons are stored, collected and transferred in digital form.

Privacy leak in this paper is also a kind of malicious behavior to invade the data privacy. It can be

described from four aspects: content, source, procedure and severity. Leak content refers to what kind of privacy data is leaked. Leak source means the storage form of privacy data. Leak procedure records the related system call sequence and the final destination of privacy data. Leak severity calculates the damage degree of such a privacy leak behavior. The details of the four aspects are as follows.

2.2 Privacy Leak Content

The content of privacy data includes many aspects such as financial privacy, Internet privacy, medical privacy, political privacy, and so on.

Financial privacy, in which the information about a person's financial transactions is guarded, is important for the avoidance of fraud or identity theft. The information about a person's purchases can also reveal a great deal of his/her history, such as the places he/she has visited, the persons he/she has contacted with, the products he/she uses, and his/her activities and habits.

Internet privacy is the ability to control what information one reveals about oneself over the Internet, and to control who can access that information. These concerns include whether emails can be stored or read by third parties without consent, or whether the third parties can track the websites someone has visited. Another concern is whether the visited websites collect, store, or possibly share personally identifiable information about users. Tools used to protect privacy on the Internet include encryption tools and anonymity services like I2P and Tor.

Medical privacy means the secrecy of a person's medical records, because of his/her insurance coverage or employment, or because he/she would not wish others to know about medical or psychological conditions or treatments that might be embarrassing. Revealing medical data could also reveal other details about one's personal life.

Political privacy has been a concern since voting systems emerged in ancient times. The secret ballot is the simplest and most widespread measure to ensure that political views are not known to anyone other than the original voter — it is nearly universal in modern democracy, and considered as a basic right of citizenship. In fact, even if other rights of privacy do not exist, the political privacy holds in most of the situations.

2.3 Privacy Leak Data Source

According to the storage type in the computer, we divide the privacy data source into the following four categories. Our later discussion will focus on privacy leak behavior from the four data sources. This classification is concluded from usual privacy leak events. Many real-world attacks focus on these four categories of privacy data. This classification is not able to cover all the categories of privacy data, but our model is easy to expand by adding new modules for other categories of privacy data.

Normal File Data Source. Most privacy data sources are stored statically as normal text file, image file, audio file and video file such as personal notes, resumes, confidential documents, photos and video records. These kinds of data sources are likely to be read and sent out.

Application Associated Data Source. Some frequently used applications such as web browser, video player, email client and address book manager can store lots of user related privacy data with special data format. These kinds of data source can be utilized and inferred to gather user favorite, website cookies, usage habit and social relationship, etc.

System Associated Data Source. System information is also an important kind of privacy data source. The basic machine information of computer, operating system profile, system configuration and user configuration are all prone to leakage, and analyzed to gather privacy information. For example, MAC address is unique to a network interface, and can therefore be used for tracking a device and its users. A machine name can also reveal personal information.

Dynamic Data Source. Another important kind of privacy data source is different from the above three kinds. They are dynamically input into the system and are not statically stored. For example, many application clients require user registration or login, and the username and the password are dynamic input from keyboard. The messages in instant message software also contain privacy data. The registration forms of many applications which include private information such as real name, gender, home address are dynamic privacy data sources too.

2.4 Privacy Leak Procedure

We divide the process of private information leaks into two kinds of procedures: unauthorized data access and covert network transmission.

Unauthorized Data Access. Unauthorized data access means two kinds of illegal data manipulation on privacy data source. One is that non-associated application accesses the associated data source, e.g., a non-web-browsing application accesses the browsing history or website cookie file. The other is that applications manipulate important system data source beyond constraint, e.g., a calculator requests network configuration.

Covert Network Transmission. Covert network transmission may take advantage of different transport protocols. We focus on raw socket, FTP and HTTP for their prevalence and easy usage. In general, PLS sends out privacy data in a similar way as benign software by firstly building a connection, and then transferring some data. But PLS can still be differentiated from benign software by mainly three features: 1) sensible data which come from unauthorized data access; 2) abnormal communication port and remote server address; 3) suspicious control command exchange.

2.5 Privacy Leak Severity

The severity of privacy leak behavior is the overall evaluation index about PLS. It refers to two factors: effect and stealth. The effect factor values the negative effect caused by PLS. It can be calculated from the importance and the amount of leaked privacy data. The stealth factor describes the resistance of PLS against the detection. It is mainly estimated by the concealing tricks used in the privacy data leak procedure.

3 Privacy Petri Net

Our analysis of PLS mainly depends on an abstract model we present called Privacy Petri Net (PPN). PPN is a high-level Petri net which focuses on PLS and has three main features. Firstly, PPN has a formal mathematical definition of syntax and semantics. This provides the precise specification of the target PLS behavior and is the foundation to define various behavior properties. Secondly, PPN has concise and powerful modeling primitives for graphical abstraction. Specific graph structures can be used to identify unique private information leak behaviors of PLS. Finally, PPN is modularized and can be used to build hierarchical models of PLS. We can use PPN to model different sub-types of privacy leaks and form complicated models. The details of the formal definition of PPN are as follows.

3.1 Basic Element

As shown in Fig.1, we define the basic element of Privacy Petri Net as follows.

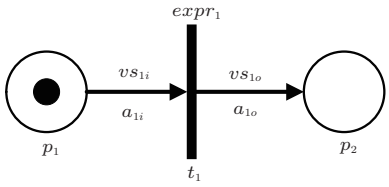


Fig.1. Basic element of Privacy Petri Net.

Definition 1 (Privacy Petri Net (PPN)). *A PPN is a seven-tuple* $(P, T, A, INST, f_{\text{pos}}, f_{\text{trans}}, f_{\text{arc}})$*, where:*

1) P *is a finite set of places. Each element in* P *denotes a local status for the subroutine of application execution.*

2) T *is a finite set of transitions and* $P \cap T \neq \emptyset$*. Each element in* T *denotes a system call or API call.*

3) $A \subseteq P \times T \cup T \times P$ *is a set of directed arcs that connect the positions and the transitions. The arcs which head to one transition are called input arcs of the transition, and the ones which emit from it are called output arcs.*

4) $INST = \{(ctg, cont, proc) : ctg \in Ctg, cont$ *and* $proc$ *are Strings*$\}$ *is the set of instance of privacy data. It is the most important element in PPN and is denoted by token with special attributes. Each token has three attributes that are related to privacy leakage.*

The first attribute denotes the four categories of data source we mentioned in Subsection 2.3*, which has some string values:*

$$Ctg = \{\text{“}File\text{”}, \text{“}Application\text{”}, \text{“}System\text{”}, \text{“}Dynamic\text{”}\}.$$

The Content attribution is different according to the Category attribute. For example, it would be the file path and file name for normal file data source, or the registry key name and key value for some kinds of system data source.

The Proc attribution denotes the steps that the token traverses through the whole PPN. It is a sequence of positions and transitions that can be denoted as:

$$Proc = p_1 t_1 p_2 \ldots p_{n-1} t_{n-1} p_n, \quad n \geqslant 1,$$

where $p_i \in P, i = 1, 2, \ldots, n$*, and* $t_j \in T$*,* $j = 1, 2, \ldots, n-1$*.*

The three attributes depict the fundamental information of the privacy leakage. All the attributes are initialized with empty value when a token is created.

5) f_{pos} *is a position function denoted by:*

$$f_{\text{pos}} : P \rightarrow \{\text{“}Start\text{”}, \text{“}Source\text{”}, \text{“}Absorb\text{”}, \text{“}Mid\text{”}, \text{“}Discrim\text{”}\}.$$

*This mapping assigns property that denotes the role of privacy leaks to each position. Different positions are indicated with distinctive icons as follows in Fig.*2.

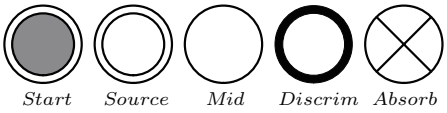


Fig.2. Position icons.

Start position is unique, which spawns new tokens with unassigned privacy attributes. Source position denotes the access point of privacy data sources. Discrim position denotes that the behavior of privacy leaks can be discriminated when a token reaches such positions. Absorb position denotes that the operation on the related data source has been checked and considered to have no privacy leak behaviors. Mid positions are all the other positions in PPN except the above four kinds of positions.

6) f_{arc} *is an arc function set that assigns a variable set to each arc, which is depicted as:*

$$f_{\text{arc}} : A \rightarrow VARSET.$$

*Each variable set VARSET mapping to input arc is called input variable set. The ones mapping to output arc is called output variable set. In Fig.*2, vs_{1i} *and* vs_{1o} *are input variable set and output variable set mapping to arcs* a_{1i} *and* a_{1o} *respectively. These variable sets can also be divided into two different categories: one is the parameter used to support the system call or API call, such as integer, float, handle, pointer, string and struct; the other is to support the privacy attributions and is mainly computed from the parameter variables.*

7) f_{trans} *is a transition function that maps an expression to each transition:*

$$f_{\text{trans}} : T \rightarrow EXPR.$$

*EXPR is a Boolean operation of some privacy leaks checking conditions that fire the related transition. In Fig.*1, *the* $expr_1$ *is the expression mapped from transition* t_1. *These conditions include many categories such as the current system call or API call name checking, current system environment variable value checking, globe system configuration checking, user operation checking and other predefined constraint conditions. There must be at least one condition that checks the current system call or API call name because we mainly depict the application behaviors with the call sequence.*

Definition 2 (PPN Module). *Assuming that m is a PPN, p is a position in its position set: m is PPN module* $\Leftrightarrow \exists p : f_{\text{pos}}(p) = \text{“}start\text{”} \wedge \exists p : f_{\text{pos}}(p) = \text{“}absorb\text{”} \wedge \exists p : f_{\text{pos}}(p) = \text{“}source\text{”} \wedge \exists p : f_{\text{pos}}(p) = \text{“}discrim\text{”}$.

PPN module is a special sub PPN with certain positions. Each PPN module must have at least one “start” position, one “absorb” position, one “source” position and one “discrim” position to depict the complete privacy leak procedure.

PPN module is defined for the privacy leak procedure to implement the modularization of PPN.

Definition 3 (Global PPN). *Global PPN is designed for the whole privacy leak behaviors:*

$$GP = (dm_1 \cup dm_2 \cup \ldots \cup dm_i) \cap (nm_1 \cup nm_2 \cup \ldots \cup nm_j),$$

where $i \geqslant 1$ *and* $j \geqslant 1$.

It contains a set of PPN modules as components. $dm_1, dm_2, \ldots, dm_j$ are PPN modules for unauthorized data access and $nm_1, nm_2, \ldots, nm_j$ are PPN modules for convert network transmission. There must be at least one *dm* and one *nm* to form a complete privacy leak behavior. $\cap$ denotes that the modules have different behavior procedure types and can be connected by transitions from the discrimination position in the former module to the start position in the later module. $\cup$ denotes that the modules have the same behavior procedure type and should not be connected.

Definition 4 (Constraint Set). *Constraint set is used to further differentiate benign application behaviors from privacy leak behaviors. The elements in constraint set are described as:*

$$cs = (rule, score_{\text{effect}}, score_{\text{stealth}}).$$

In some occasions, it is difficult to judge the application behaviors with only the position-transition sequence triggered by system call name. Therefore, the system call parameters and the return value are considered. Each rule is about the parameters and the return value that benign applications should obey. The other two scores, effect and stealth, are used to calculate the severity of the privacy leak behavior which we will discuss later. Higher effect score means privacy information is more important, while high stealth score means that the leak behavior is well disguised.

3.2 Main Operation

We now describe the work mode of PPN. PPN contains four main operations that change the status of positions, arcs, transitions and tokens.

Definition 5 (Privacy Instance Spawning and Removing). *Privacy instance spawning is the operation on PPN positions denoted as:*

$$SPAWN(p,\ inst),$$

where $f_{\text{pos}}(p) =$ "*start*", *inst is a new privacy instance.*

As defined above, token denotes the instance of privacy data source. Spawning of privacy instance is the operation that creates a new token in the *start* position with empty attributions. After the target application starts running, whenever the *start* position is empty, a new token is spawned in the *start* position with empty attributions.

Privacy instance removing is also the operation on PPN positions:

$$REMOVE(p,\ inst),$$

where $f_{\text{pos}}(p) =$ "*absorb*" and *inst*, a privacy instance in p.

If the token arrives at a discrimination position and all the privacy attributions are assigned, then the leak procedure of this privacy data source is discovered and recorded completely. However, if the token finally stops in an absorb position, it is discarded and removed from the PPN.

Definition 6 (Privacy Variable Binding). *Privacy variable binding is the operation that assigns values to the variables in the variable set of the target arc a:*

$$BIND(a,\ valueset),$$

where $\forall x \in f_{\text{arc}}(a) : x$ *has been assigned, valueset contains values for the related variables in VARSET of a.*

Privacy variable binding means two kinds of variable assignment. The first one is to give a value to each free variable appearing in the input variable set. These values are mainly obtained from two sources: one is the parameters of call sequence of target application, the other is the output variable set of the previous transition. The second assignment is for the free variables in the output variable set. These values are mainly used for computing the attributes in the privacy instance denoted by tokens and are generated after privacy leak condition triggering which will be defined in Definition 7.

Definition 7 (Privacy Leak Condition Triggering). *Privacy leak condition triggering is the operation that moves the tokens from position to position through target transition t:*

$$TRIGGER(t, inst_1, inst_2,\ \ldots,\ inst_i),$$

where $\forall e \in f_{\text{trans}}(t) : e$ *is true and* $inst_i$ $(i \geqslant 1)$ *is the token in the position connecting to the position t.*

A transition can occur when all the variables appearing in the input variable set are assigned and all the conditions defined by transition function f_{trans} are satisfied. After the occurring of a transition, the tokens in the positions that connect to it will move into the positions that connect from it. The variable values in the output variable set are also used to compute the privacy attributes of the tokens.

3.3 Discrimination Theorems

We give some important theorems of PPN for checking the privacy leak behaviors.

Definition 8 (Behavior Path Set). *Assuming that* $C = c_1c_2 \ldots c_n$ *is the call sequence of the target application,* $inst_1, inst_2,\ \ldots,\ inst_m$ *are the privacy instances which are spawned when the calls in C are checked one by one to trigger the transitions. Then the behavior path set BPS is the union of move trace of all the privacy instances:*

$$BPS = \{inst_1.proc,\ inst_2.\ proc,\ \ldots,\ inst_m.proc\}.$$

Behavior path set BPS is the mapping from the behavior of the target application to the move traces of tokens in PPN. Each call sequence C can be mapped to one unique BPS to describe the behavior of privacy leaks in all the behaviors of the target application.

Definition 9 (Leak Path and Leak Reachability). *Leak path is a special privacy instances' move traces in behavior path set BPS. Let* $inst_k.proc = p_1t_1p_2 \ldots p_{n-1}t_{n-1}p_n$ *and* $inst_k.proc \in BPS$: $inst_k.proc$ *is a leak path* $\Leftrightarrow$ *the privacy instance* $inst_k$ *has leak reachability* $\Leftrightarrow \exists i, j:\ f_{\text{pos}}(p_i) =$ "*source*" $\wedge$ $f_{\text{pos}}(p_j) =$ "*discrim*".

A leak path must have one source position and one discrimination position. This definition confirms that a private instance can move along this path and leak the private data in source position.

With the leak reachability property, we can verify the target application about whether certain type of private information is leaked, as shown in Theorem 1.

Theorem 1. *If a privacy instance inst spawned in a PPN module m is with leak reachability, then the target application's call sequence C contains privacy leak behaviors procedure modeled by the PPN module.*

Proof. Since *inst* is a privacy instance with leak reachability, by Definition 8, $inst.proc = p_1t_1p_2 \dots p_{n-1}t_{n-1}p_n$ is a leak path and $\exists i, j$: $f_{\text{pos}}(p_i)$ = *"source"* $\wedge$ $f_{\text{pos}}(p_j)$ = *"discrim"*. In addition, *inst* is spawned in m, and by Definition 9, m has at least one source position and one discrimination position; hence, *inst.proc* contains a sub path $proc' = p_it_ip_{i+1} \dots p_{j-1}t_{j-1}p_j \wedge p_i \in m \wedge p_j \in m$. Because by Definition 8, $inst.proc \in BPS$ and it is created by call sequence C, and by Definition 1, each transition has at least one condition which checks the current call name, each t_i has a related call c_i, and $\exists C' = c_ic_{i+1} \dots c_{j-1}c_j \wedge \forall k, i \leqslant k \leqslant j, c_k \in C, C'$ can trigger the transitions on *proc*' in the PPN module m. This sub-call sequence C' forms a privacy leak behaviors procedure modeled by m. Therefore, Theorem 1 holds. □

Theorem 2. *Let a privacy instance $inst_a$ be spawned in an unauthorized data access PPN module dm_1 and another privacy instance $inst_b$ be spawned in a covert network transmission module nm_1. If*

1) *$inst_a$ is with leak reachability,*

2) *$inst_b$ is with leak reachability,*

3) *$inst_a.Cont \in inst_b.Cont$;*

then target application's call sequence C contains complete privacy leak behavior modeled by global PPN $gp_1 = dm_1 \cap nm_1$.

Proof. By Theorem 1, since 1), $\exists C'_a \in C, C'_a$ can trigger the transitions on $inst_a.proc$ in the PPN module dm_1; since 2), $\exists C'_b \in C, C'_b$ can trigger the transitions on $inst_b.proc$ in the PPN module nm_1. By Definition 3, since $gp_1 = dm_1 \cap nm_1$, assuming that the discrimination position in $inst_a.proc$ is p_a and the start position in $inst_b.proc$ is p_b. Since 3), the two sub behavior procedures have consistent content, let a new transition t_0 connect from p_a to p_b for simply transporting the output value set of p_a to the input value set of p_b, and c_0 is an abstract related call of t_0. Hence there will be a new leak path $p_1t_1 \dots p_at_0p_b \dots t_{n-1}p_n$ through the two modules and the call sequence $C'_ac_0C'_b$ forms a complete privacy leak behaviors modeled by gp_1. Therefore, Theorem 2 holds. □

3.4 Privacy Leak Calculating Rules

At last, we give the definition of privacy leak model which is the core of our approach to analyze privacy leaks by PPN.

Definition 10 (Privacy Leak Behavior). *We formulize the privacy leak behavior with the four-tuple $PL(p_{\text{ctg}}, p_{\text{cont}}, p_{\text{proc}}, p_{\text{severity}})$. $p_{\text{ctg}}, p_{\text{cont}}, p_{\text{proc}}$ are similar to the three privacy attributes of the tokens in Definition* 1 *but are related to the whole PPN rather than a single token. p_{severity} is used to judge the damage degree of leak behavior and is calculated from the constraint set in Definition* 4*:*

$$p_{\text{severity}} = \left(\sum_i cs_i.score_{\text{effect}}, \sum_i cs_i.score_{\text{stealth}} \right),$$

where cs_i is the element in the constraint set.

The privacy leak four-tuple *PL* is then mapped from the target application *app* with PPN. It is calculated according to the static features of PPN including the position property, the variable set of arcs and the transition expression. And the final output value is computed with the execution of the target application and the privacy attributes of the tokens.

There are three calculating rules defined for PPN: global rule, module rule and path rule. The global rule evaluates the whole privacy leak behaviors. The module rule is defined for the sub-procedures of leak behaviors. The path rule is defined for the single leak path.

Rule 1 (Path PL Calculating). *By Theorem* 1*, the path PL is calculated by the privacy instance moving through the path. Assume that $path_1 = p_1t_1p_2 \dots p_{n-1}t_{n-1}p_n$ that connects a source position and a discrimination position is a leak path, and their transition expressions are denoted by $e_i, 1 \leqslant i \leqslant n-1$. If a private instance $inst_1$ has leak reachability on $path_1$, then all the transitions on this path must be triggered. Thus the sub privacy leak tuple of this path is computed:*

$$PL_{\text{path}_1} = (inst_1.Ctg, inst_1.Cont, inst_1.Proc, P_{\text{severity}}),$$

where $E_1 = e_1 \wedge e_2 \wedge \dots \wedge e_i$ is true. P_{severity} is the severity computed by the constraints broken by $path_1$.

Rule 2 (Module PL Calculating). *The module PL is computed from all the possible leak paths. If there are j different paths in the PPN module m_1, the module privacy leak tuple can be computed by:*

$$PL_{m1} = PL_{\text{path}_1} \vee PL_{\text{path}_2} \vee \dots \vee PL_{\text{path}_j},$$

where $E_m = E_1 \vee E_2 \vee \dots \vee E_j$ is true.

The conjunction symbol from PL_{path_1} to PL_{path_j} describes the union of elements of the tuples.

Rule 3 (Global PL Calculating). *By Theorem* 2*, there should be at least one leak path in unauthorized*

data access module and another leak path in covert network transmission module. Therefore, the global PL is computed as follows:

$$PL_{\text{global}} = (PL_{da_1} \vee PL_{da_2} \vee \ldots \vee PL_{da_N}) \wedge (PL_{ct_1} \vee PL_{ct_2} \vee \ldots \vee PL_{ct_M}).$$

PL_{global} is the global privacy leak tuple that gives the output of the entire leak behavior. It is computed by the privacy leak tuples of the modules da_1 to da_N which denote the unauthorized data access and the modules ct_1 to ct_M which denote the covert network transmission. Since there may be many different leak data sources in one application, we use conjunction symbol between da_1 to da_N and ct_1 to ct_M to describe the union of items such as "p_{ctg}" and "p_{cont}" in the output value of privacy leak tuple. However, by Theorem 2, there must be a composition of at least one data source access procedure and one covert network transmit procedure to form a complete privacy leak behavior, and we therefore use disjunction symbol to describe the concatenation of "p_{proc}" between the two kinds of procedures. "p_{severity}" is the sum of all the module severity scores.

4 Modeling PLs with PPN

In this section, we first build the global PPN for the whole privacy leak behaviors, and then build some PPN modules for typical privacy leak procedures.

4.1 Global PPN for Whole Privacy Leak Behaviors

According to Definition 3, global PPN consists of unauthorized data access modules and covert network transmission modules. It can be expanded easily by inserting new modules to enhance its modeling ability.

As shown in Fig.3, in global PPN *gp*, we consider four kinds of data access modules: normal file, application data, system data and dynamic data. We further consider three kinds of network transmission modules: raw socket connection, FTP connection and HTTP connection. All other details except start position, source position and discrimination position are omitted and will be discussed later. According to Theorem 2, these modules are connected by a special transition t_0 to form complete privacy leak behavior. This t_0 is not a transition for a single system call or API call, but for any correlation between the output value set of data access modules and the input value set of network transmission modules.

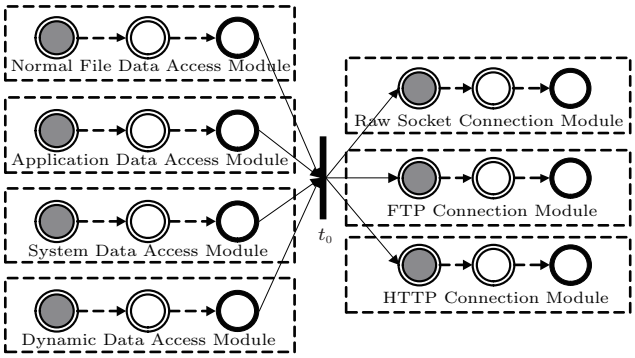


Fig.3. Global PPN *gp* which contains seven modules.

Next, we build PPN modules for different privacy leak behavior procedures and discuss modeling applications on Windows platform. We first give the graphic presentation, then summarize the leak behavior procedures and find the possible leak paths. Finally, we present some details of positions, transitions and arcs.

4.2 PPN Modules for Unauthorized Data Access

We first give PPN modules unauthorized data access. There are four kinds of data access modules: normal file, application data, system data and dynamic data.

4.2.1 PPN Module for Normal File Data Access

In Fig.4, to access data in a normal file, the target application should first check the possible directories to find the file by "NtQueryDirectoryFile" or "NtNotifyChangeDirectoryFile", then achieve the file handle by "NtCreateFile" or "NtOpenFile", and at last get the file properties or read the file content by "NtFsControlFile" or "NtReadFile". The details of this module are shown in Table 1. There are three possible leak

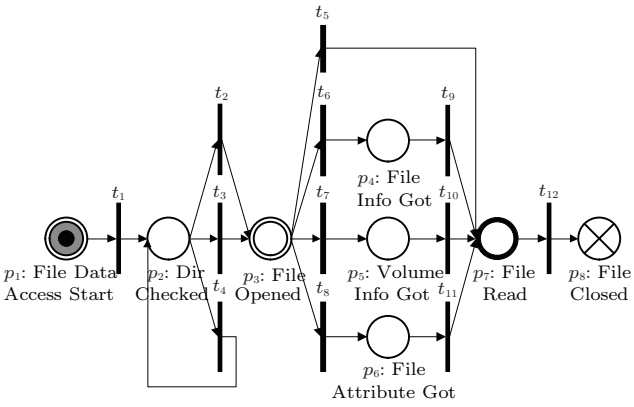


Fig.4. Module m_{nfda}: PPN module for normal file data access.

Table 1. Details of Module m_{nfda}

	EXPR	Input Variable Set	Output Variable Set
t_1	"NtQueryDirectoryFile"	Dir name	"Ctg"
t_2	"NtCreateFile"∧ Path Check	File name	File handle
t_3	"NtOpenFile"∧ Path Check	File name	File handle
t_4	"NtNotifyChangeDirectoryFile"	Dir name	File handle
t_5	"NtFsControlFile"	Ctl Code	Pointer, "Cont"
t_6, t_7, t_8	"NtQueryInformationFile" "NtQueryAttributeFile" "NtQueryVolumeInformationFile"	File handle	Pointer, "Cont"
t_9, t_{10}, t_{11}	"NtReadFile"	File handle	Pointer, "Cont"
t_{12}	"NtClose"	File handle	NULL

paths: $p_3t_6p_4t_9p_7, p_3t_7p_5t_{10}p_7$ and $p_3t_8p_6t_{11}p_7$. To differentiate from the legitimate file access, we check the file path for further verification. Only application installation path, system path and application creation path are permitted to be accessed.

4.2.2 PPN Module for Application Data Access

As depicted in Fig.5, application data includes many categories and almost each frequently-used application has its own privacy-related dataset. We mainly discuss IE kernel web browser here as an example. The browser contains lots of private data: browser page history can be leaked by "EnumUrls" interface of "IUrlHistoryStg" object, favorite pages can be revealed by "ImportExportFavorites" of "IShellUIHelper" object, typed URL history can be got from register key "Software\\Microsoft \\Internet Exploer\\TypedURLs", website cookies can also be leaked from "InternetGetCookie" API, and even protected cookies may be obtained by "IEGetProtectedModeCookie". The details of this module are shown in Table 2. There are five possible leak paths: $p_2t_5p_5, p_2t_6p_6, p_1t_2p_3, p_1t_3p_3, p_4t_8p_7$. The start position p_1 is also a source position. We need to check the application itself to confirm leak behaviors. Only web browser and other browsing tools can invoke such API and system calls.

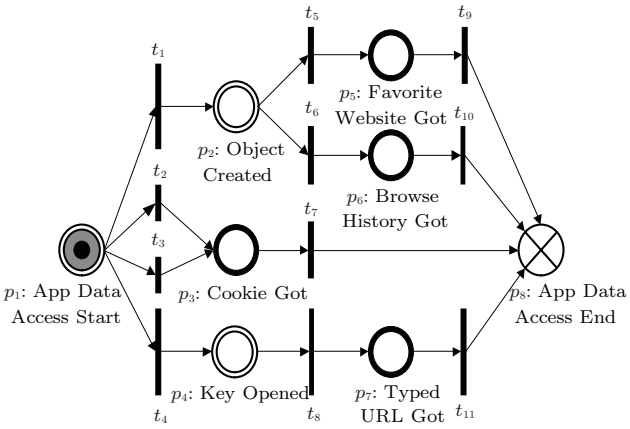


Fig.5. Module m_{ada}: PPN module for application data source.

4.2.3 PPN Module for System Data Access

As depicted in Fig.6, system data is about basic system statuses and configurations such as basic computer info, network info and user info. We only

Table 2. Details of Module m_{ada}

	EXPR	Input Variable Set	Output Variable Set
t_1	"CoCreateInstance" ∧ object name check	Object name	"Ctg", COM Obj
t_2	"InternetGetCookie" ∧ application check	URL name, cookie name	"Ctg", "Cont", cookie value
t_3	"IEGetProtectedModeCookie" ∧ application check	URL name, cookie name	"Ctg", "Cont", cookie value
t_4	"NtOpenKey" ∧ application check	Key name	"Ctg", key handle
t_5	"EnumUrls"	COM object	"Cont", URL list
t_6	"ImportExportFavorites"	COM object	"Cont", favorites
t_7	"NtDeleteFile"	File handle	NULL
t_8	"NtQueryValueKey"	Key handle	"Cont", keyvalue
t_9, t_{10}	"CoUninitialize"	COM object	NULL
t_{11}	"NtClose"	Key handle	NULL

list frequently-used calls here, such as "GetWindowsDirectory" or "GetVersion" for basic computer info, "GetAdaptersInfo" or "GetNetworkParams" for local network info and "NetUserEnum" for use info. The details of this module are shown in Table 3. All paths which connect p_1 and p_2, p_3, p_4 are leak paths.

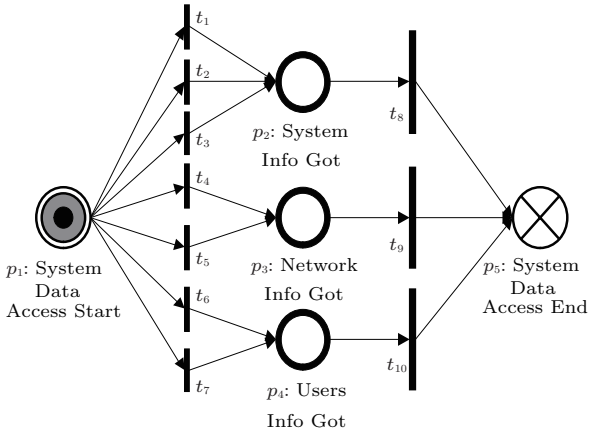


Fig.6. Module m_{sda}: PPN module for system data access.

Table 3. Details of Module m_{sda}

	EXPR	Input Variable Set	Output Variable Set
t_1	"GetWindowsDirectory"	String pointer	"Ctg", "Cont"
t_2	"GetVersion"	String pointer	"Ctg", "Cont"
t_3	"GlobalMemoryStatus"	Struct pointer	"Ctg", "Cont"
t_4	"GetAdaptersInfo"	Struct pointer	"Ctg", "Cont"
t_5	"GetNetworkParams"	Struct pointer	"Ctg", "Cont"
t_6	"GetUserName"	String pointer	"Ctg", "Cont"
t_7	"NetUserEnum"	Struct pointer	"Ctg", "Cont"
t_9, t_{10}, t_{11}	"NtClose"	Handle	NULL

4.2.4 PPN Module for Dynamic Data Access

In Fig.7, for dynamic data access, we mainly consider keyboard and mouse input. There are two methods to get keyboard and mouse input: use the hook function by first setting hook by "SetWindowsHook", then use the keyboard or mouse callback function, then release the hook; or get window handle by "GetActiveWindow", then get key stroke by "GetKeyState" or "GetAsyncKeyState" and get mouse state by "GetMouseMovePoints". The details of this module are shown in Table 4. There are four possible leak paths: $p_2t_3p_4t_7p_8, p_2t_4p_5t_8p_9, p_3t_5p_6t_9p_{10}, p_3t_6p_7t_{10}p_{10}$.

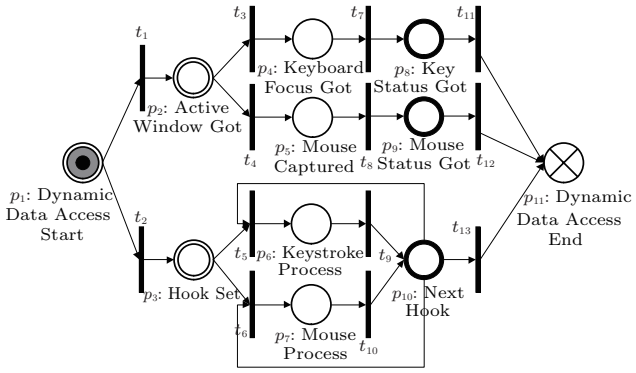


Fig.7. Module m_{dda}: PPN module for dynamic data access.

4.3 PPN Modules for Covert Network Transmission

We give PPN modules for covert network transmission in this subsection. We consider three kinds of connections: socket connection, FTP connection and HTTP connection.

4.3.1 PPN Module for Socket Connection

In Fig.8, to build socket connection, the target application should first create a socket by "socket", and then build server side by "listen" or client side by "connect". The application can now transfer privacy data by "send" or "sendto", and receive remote data by "recv" or "recvfrom". After the data transfer phase, the application ends the process by "closesocket". The details of this module are shown in Table 5. There are three possible leak paths: $p_2t_3p_3t_6p_5t_8p_6, p_2t_3p_3t_7p_5t_8p_6, p_2t_2p_4t_5p_6$.

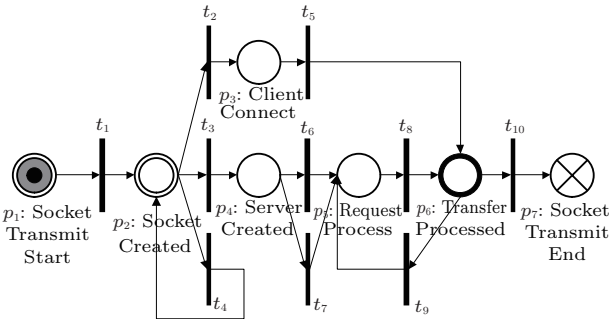


Fig.8. Module m_{socket}: PPN module for socket connection.

4.3.2 PPN Module for HTTP Connection

HTTP connection can be implemented by two function libraries: WinInet and WinHTTP. WinINet is widely used for client building and WinHTTP has

Table 4. Details of Module m_{dda}

	EXPR	Input Variable Set	Output Variable Set
t_1	"GetActiveWindow"	NULL	"Ctg", handle
t_2	"SetWindowsHook"	Hook type, hook proc	"Ctg", hook handle
t_3	"GetFocus"	NULL	Window handle
t_4	"GetCapture"	NULL	Window handle
t_5	"KeyboardProc"	Keystroke info	"Cont", key status
t_6	"MouseProc"	Mousemove info	"Cont", mousemove
t_7	"GetKeyState"	Key name	"Cont", key status
t_8	"GetMouseMovePoints"	Struct pointer	"Cont", mousemove
t_9, t_{10}	"CallNextHook"	Hook name	Next hook handle
t_{11}	"Close"	Window handle	NULL
t_{12}	"ReleaseCapture"	Window handle	NULL
t_{13}	"UnSetWindowsHook"	Hook handle	NULL

Table 5. Details of Module m_{socket}

	EXPR	Input Variable Set	Output Variable Set
t_1	"Socket"	Address type, protocol	Socket handle
t_2	"Connect"	Socket handle, Sockaddr	"Dest"
t_3	"Listen"	Socket handle, Sockaddr	"Dest"
t_4	"Bind"	Socket handle, Sockaddr	"Dest"
t_5, t_8	"Send" $\vee$ "Sendto"	Socket handle, buff pointer	"Cont", status code
t_6	"Accept"	Socket handle, Sockaddr	Status code
t_7	"Select"	Socket handle, Sockaddr	Status code
t_9	"Recv" $\vee$ "Recvfrom"	Socket handle, buff pointer	"Cont", status code
t_{10}	"Closesocket"	Socket handle	NULL

http server implementation. For building HTTP connection and leaking privacy data, the target application firstly creates Internet handle by "InternetOpen", then sets up the connection to remote HTTP server by "InternetConnect", and then creates HTTP request by "HttpOpenRequest" adding HTTP headers by "HttpAddRequestHeader". Finally the application sends data by post HTTP request with "HttpSendRequest" or directly writes Internet file by "InternetWriteFile", as depicted in Fig.9. The details of this module are shown in Table 6. Two leak paths to send data are $p_2t_2p_3t_3p_4t_6p_6$ and $p_2t_2p_3t_3p_4t_4p_5t_7p_7$.

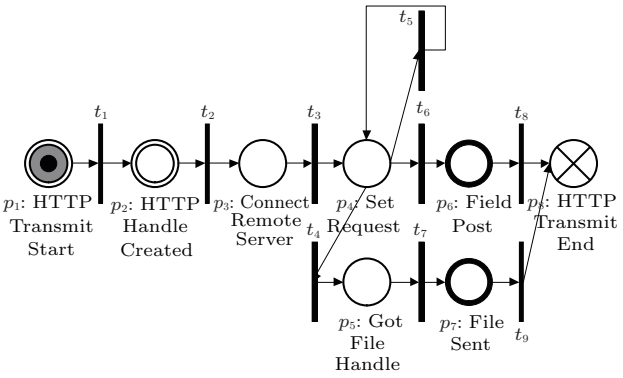


Fig.9. Module m_{HTTP}: PPN module for HTTP connection.

4.3.3 PPN Module for FTP Connection

As shown in Fig.10, FTP connection is used to transfer privacy data in files, which is especially useful for transferring a large number of files or a large file in size. In windows API, FTP service is also implemented in WinINet library and the leak procedure is similar to HTTP connection. An Internet handle and connection is to be created. The application then sets file transferring directory by "FtpSetCurrentDirectory" or "FtpCreateDirectory". Then, the privacy data are sent by "FtpPutFile" directly in files, or the data

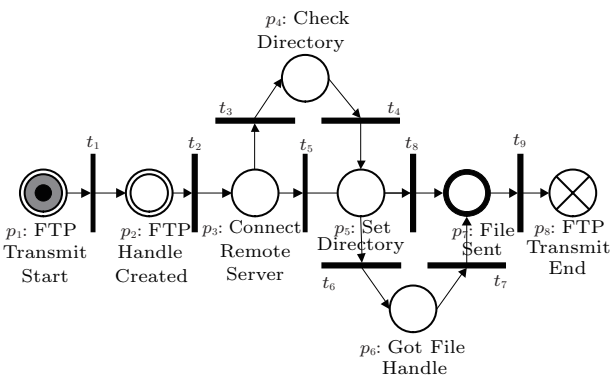


Fig.10. Module m_{FTP}: PPN module for FTP connection.

Table 6. Details of Module m_{HTTP}

	EXPR	Input Variable Set	Output Variable Set
t_1	"InternetOpen"	Access type, proxy info	"Dest", Internet handle
t_2	"InternetConnect"	Internet handle, server info	"Dest", Internet handle
t_3	"HttpOpenRequest"	Internet handle	"Cont", Internet handle
t_4, t_6	"HttpSendRequest"	Internet handle, request header	"Cont", status code
t_5	"HttpAddRequestHeader"	Internet handle, request header	"Cont", Internet handle
t_7	"InternetWriteFile"	Internet handle, buff pointer	"Cont", status code
t_8, t_9	"InternetCloseHandle"	Internet handle	NULL

Table 7. Details of Module m_{FTP}

	EXPR	Input Variable Set	Output Variable Set
t_1	"InternetOpen"	Access type, proxy info	"Dest", Internet handle
t_2	"InternetConnect"	Internet handle, server info	"Dest", Internet handle
t_3	"FtpGetCurrentDirectory"	Internet handle, string pointer	Status code
t_4	"FtpSetCurrentDirectory"	Internet handle, dir name	"Cont", status code
t_5	"FtpCreateDirectory"	Internet handle, dir name	"Cont", status code
t_6	"FtpOpenFile"	Internet handle, file name	"Cont", file handle
t_7	"InternetWriteFile"	Internet handle, buff pointer	"Cont", status code
t_8	"FtpPutFile"	Internet handle	Status code
t_9	"InternetCloseHandle"	Internet handle	NULL

are to be written to FTP file handle by "FtpOpenFile" and "InternetWriteFile". The details of this module are shown in Table 7. Two main leak paths are $p_2t_2p_3t_5p_5t_8p_7, p_2t_2p_3t_5p_5t_6p_6t_7p_7$.

5 Analyzing PLS with PPN

We first describe our analysis framework for PLS based on PPN and then present the related algorithms.

5.1 Approach Framework

Our framework includes three main components: call tracer, PPN module library and privacy leaks analyzer. The details are as follows.

As shown in Fig.11, PLS instance runs in the virtual OS. First the call sequence is captured by the call tracer. Then the filtered call sequence is sent to the analyzer for generating PPN instance with the related PPN modules from the module library. After the privacy leaks analyzer checks the leak path and calculates the privacy leak tuple, the analysis result is output as the four-tuple we defined in Definition 10.

5.1.1 Call Tracer

The call tracer mainly focuses on the system call and API call of the application instance. We use hook functions to intercept important calls which are related to the privacy leak behavior of application. To get more precise information and build more accurate model, we not only get the call name, but also obtain the parameters and return values. Details of the sequence of call are recorded into log files.

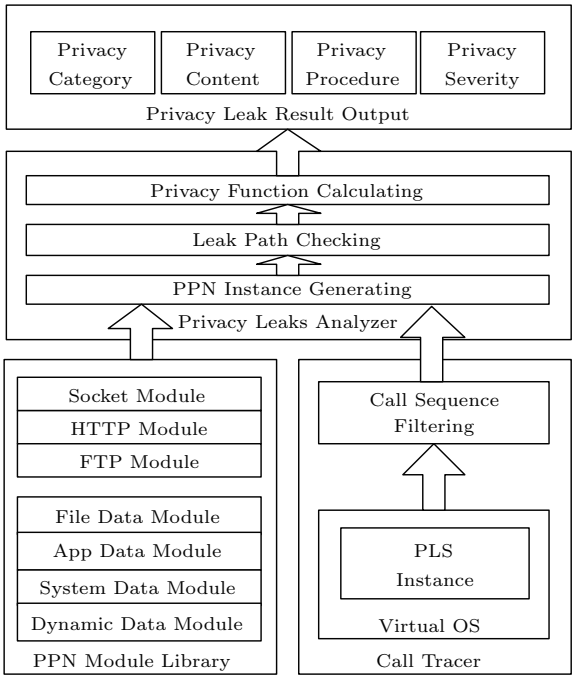


Fig.11. Analysis framework of private information leaks.

5.1.2 PPN Module Library

The PPN module library stores all the PPN modules we build in Section 4: modules for unauthorized data access and modules for covert network transmission. New modules can be simply added to the library.

Such modularization of PPN enables the flexibility and scalability for PLS analysis.

5.1.3 PPN Analyzer

This component is the core of our framework. The component consists of three parts: PPN instance generating, Leaks path checking and privacy leak tuple calculating. Firstly, new PPN instances are generated from PPN module library for PLS that is considered suspicious, and then the filtered call sequence is used to spawn new token or trigger the transitions in the PPN instance as we defined in Definitions 5∼7. Secondly, all the privacy instances which arrive at discrimination position are checked to find the possible leak paths. Thirdly, privacy leak tuples are calculated from path PL tuple to module PL tuple and then global PL tuple as we defined in Definition 10.

5.2 Analyzing Algorithm

After building the PPN model, we can analyze the behavior of the target application to answer the four questions about privacy data leaks we mentioned in Section 1. We have made four algorithms to accomplish this goal.

In Algorithm 1, the call sequence gathered from runtime execution tracer and network traffic sniffer is filtered to keep only the calls that we care about. Algorithm 2 uses parameters in call sequence to do BIND operation to the variable set defined in arc function f_{arc} and checks the conditions defined in transition function f_{trans} in order to do TRIGGER operation to transitions. Algorithm 3 collects all move traces of tokens as *BPS* in Definition 8 and finds all the leak paths. Algorithm 4 synthesizes the privacy instance behavior which has leak reachability and computes the final output about the privacy leak behaviors of the target application.

Algorithm 1. Call Sequence Filtering

procedure *Filter* (C; F)
input
$C = c_1c_2 \ldots c_m$: call sequence of target application
$F = \{f_1, f_2, \ldots, f_n\}$: set of filter conditions
output
C': filtered set of call sequence
begin
  $C' := \emptyset$
  for $i := 1$ **to** m
    if $(f_1 \wedge f_2 \ldots \wedge f_n)$
      $C' := C' + c_i$
  return C'
end

Algorithm 2. PPN Instance Generating

procedure *Generate* (C'; M)
input
$C' = c_1c_2 \ldots c_m$: filtered call sequence
$c_i = (callname, params, results)$: call object
callname : name string
params : input parameter set
results : output results
$M = \{m_{\text{nfda}}, m_{\text{ada}}, m_{\text{sda}}, m_{\text{dda}}, m_{\text{socket}}, m_{\text{HTTP}}, m_{\text{FTP}}\}$: n PPN modules
vars
a_{ki}: input arc of transition t_k
a_{ko}: output arc of transition t_k
p_{ki}: the position that a_{ki} emits from
p_{ko}: the position that a_{ko} heads to
output
INST : privacy instance set
begin
  {Initialize private instance in all start positions}
  for $i := 1$ **to** n
    for p_j **in** $m_i.P$
      if $f_{\text{pos}}(p_j)$ = "start"
        $inst_1$:= **new** *inst*
        SPAWN($inst_1, p_j$)
        $inst_1.proc := p_j$
        $INST := INST \cup \{inst_1\}$
  {Check each call to activate PPN operations}

  for $i := 1$ **to** m
    for $j := 1$ **to** n
      if BIND($a_{ki}, c_i.params$) **then**
        if $(c_i \in m_j.\text{T} \wedge t_k.expr)$
          TRIGGER(t_k)
          if "Ctg" $\in a_{ko}.varset$
            $inst_k.Ctg := c_i.results.Ctg$
          if "Cont" $\in a_{ko}.varset$
            $inst_k.Cont := c_i.results.Cont$
          if "Dest" $\in a_{ko}.varset$
            $inst_k.Dest := c_i.results.Dest$
            $inst_k.proc := inst_k.proc + t_kp_{ko}$
          BIND($a_{ko}, c_i.results$)
          if $f_{\text{pos}}(p_{ki})$ = "start"
            $inst_{k+1}$:= **new** *inst*
            SPAWN($inst_{k+1}, p_{ki}$)
            $inst_{k+1}.proc := p_{ki}$
            $INST := INST \cup \{inst_{k+1}\}$
  {Remove the instances trapped in absorb position}
  for *inst* **in** *INST*
    if exists $p_i \in inst.proc \wedge f_{\text{pos}}(p_i)$ = "absorb"
      $INST := INST - \{inst\}$
  return *INST*
end

Algorithm 3. Leaks Path Checking

procedure *Search* (*INST*)
input
INST : private instances set
output
INST' : private instances which have leak reachability
begin
  INST' := $\emptyset$
  for *inst* **in** *INST*
    if exists $p_i \in inst.proc \wedge f_{\text{pos}}(p_i)$ = "source" $\wedge$
      exists $p_i \in inst.proc \wedge f_{\text{pos}}(p_i)$ = "discrim"
      INST' := *INST'* $\cup \{inst\}$
  return *INST'*
end

Algorithm 4. Privacy Leak Tuple Calculating

procedure *Synthesize* (*INST*; M)
input
INST : private instances which have leak reachability
$M = \{m_{\text{nfda}}, m_{\text{ada}}, m_{\text{sda}}, m_{\text{dda}}, m_{\text{socket}}, m_{\text{HTTP}}, m_{\text{FTP}}\}$
vars
$FP = \{fp_1, fp_2, \ldots, fp_n\}$: path privacy leak tuple set
$FM = \{fm_{\text{nfda}}, fm_{\text{ada}}, fm_{\text{sda}}, fm_{\text{dda}}, fm_{\text{socket}}, fm_{\text{HTTP}}, fm_{\text{FTP}}\}$: module privacy leak tuple set for modules in M_{da} and M_{ct}
output
Ctg : category of privacy leak behaviors
Cont : content of privacy leak behaviors
Proc : procedure of privacy leak behaviors
Dest : destination of privacy leak behaviors
begin
{Compute the PL tuple for each leak path}
for $inst_i$ **in** *INST*
$fp_i = (inst_i.Ctg, inst_i.Cont, inst_i.Proc, inst_i.Dest)$
{Compute the PL tuple for each module}
for fm_j **in** *FM*
for fp_i **in** *FP*
if $inst_i \in m_j$
$fm_j := fm_j \vee fp_i$
{Compute the global PL tuple}
for fm_i **in** $\{fm_{\text{nfda}}, fm_{\text{ada}}, fm_{\text{sda}}, fm_{\text{dda}}\}$
for fm_j **in** $\{fm_{\text{socket}}, fm_{\text{HTTP}}, fm_{\text{FTP}}\}$
if $fm_i.Cont \in fm_j.Cont$
$fg := fg \vee (fm_i \wedge fm_j)$
return *fg.Ctg, fg.Cont, fg.Proc, fg.Severity*
end

Filtering is the first step to process the call sequence. Each call in call sequence C will be checked by all the conditions in F. These conditions are divided into two kinds: one is to check whether the call name is in our PPN modules, the other is to consider whether the parameters are related to our definition in variable sets. Since the amount of call sequence is quite large, this step will remove irrelevant calls and decrease the complexity of consequent analyzing steps. The time complexity is $O(mn)$ which is mainly related to the length m of call sequence C and the size n of filter condition set F.

PPN instance generating procedure follows the PPN module library and uses the filtered call sequence from Algorithm 1 and the modules we defined in Section 4 as the input. At the beginning, all the start positions spawn one empty privacy instance. For each call in C', we first find the related transition from all the modules and bind the input arc with the input parameters, then trigger the transition and move the token to the next position. Then the privacy attribute is calculated and the call results are to be bound to the output arc. A new instance is spawned in the start position. At the end, the instances trapped in absorb position are removed. The time complexity is $O(mn)$ which is mainly related to the length m of filtered call sequence C' and the size n of PPN module set M.

After generating the PPN instance and activating PPN with the call sequence, all the possible privacy instances that can have leak reachability are collected in *INST*. According to Theorem 1, the privacy instance we look for must have a leak path in its *proc* attribute. We search in *INST* and find all the leak reachable privacy instances. The time complexity is $O(n)$ which is mainly related to the size n of *INST.*

According to the definition of PL tuple, the global PL tuple is computed by the module PL tuple, and the module PL tuple is computed by the path PL tuple. Therefore, we first compute all the path PL tuple output value from all the reachable privacy instances by rule 1 we defined in Subsection 3.4. We then calculate module PL tuple output value from the path leak tuples by rule 2. There must be a pair of privacy instances in both the data access module and the covert transmission module, and according to rule 3, we can therefore check all the combinations between the four data access modules $\{fm_{\text{nfda}}, fm_{\text{ada}}, fm_{\text{sda}}, fm_{\text{dda}}\}$ and the three network transmission modules $\{fm_{\text{socket}}, fm_{\text{HTTP}}, fm_{\text{FTP}}\}$. Finally, we calculate the output value *fg.Ctg, fg.Cont, fg.Proc, fg.Dest* of global PL tuple to answer the four questions we posed in Section 1. The time complexity is $O(mn)$ which is mainly related to the size m of PPN module set *FM* and the size n of instance set $INST$.

6 Real-World PLS

To understand the details of private information leaks and answer the four questions we put forward, we collect some real-world applications to apply our PPN-based approach. We first give an overview of our experimental environment and our application set, and then show some case studies of different privacy leak behaviors. We discuss the experimental results in the end of Section 6.

6.1 Experimental Settings

The experimental environment is designed on the basis of the architecture we presented in Section 3. We use VMware Workstation 7.0 as the virtual machine platform to build Windows OS image. The functionality of runtime execution tracer is undertaken by API

monitor2 r9. Wireshark 1.6.4 is used as network traffic sniffer to capture the details of privacy information leak destination. The core algorithm of PPN model is implemented by Python 3.1. All the software samples are installed and tested in different snapshots of the virtual host to avoid mutual interference.

Our experiment software sample set contains two kinds of applications.

1) Malware announced by anti-virus software contains privacy leak behaviors. We choose malware samples set from VXheaven① which has collected more than two million malware corpses. These samples contain prevalent trojan, worm, virus and other kinds of malware such as backdoor, exploiter, rootkit and hacktool. The fundamental functionality of our approach is analyzing malicious applications. Although anti-virus tools can identify these applications as malware, our approach can collect more details about their privacy leak behaviors.

2) Suspicious applications which are prevalent on free software download website Download②. These applications contain anti-virus software, multimedia tools (such as player, viewer and manipulator), instant messaging tools, download tools, web tools (such as browser and email client) and desktop utilities. Popular applications are basically safe and not likely to contain privacy leak behaviors. Therefore most of the anti-virus tools based on white-list are difficult to identify the suspicious behaviors of these applications. Our approach can fill in the gap.

Next we give experiments on each type of application in a case study way, and show more details about their privacy leak behaviors. We named the selected cases in a popular applications set with symbols but not their real name. This is not only for simplicity reasons, but also in order to keep the privacy of the software vendor. The malicious and suspicious applications are named Mi and Ai.

6.2 Malicious Applications Case Study

We first choose one typical malware sample for each kind of data source type and describe in more details about its privacy leak behaviors.

6.2.1 Normal File Data Source

M1 is a variant of Ldpinch family. This malware is a trojan-PSW which is designed to collect privacy information from local files. The call sequence fragments are depicted in Fig.12.

Fig.12. Call sequence fragment of M1. (a) M1 creates file handle with "NtOpenFile" in Fig.2. (b) M1 reads the local file with "NtReadFile" in Fig.2. (c) M1 creates the Internet handle with "InternetOpen" in Fig.7. (d) M1 creates Http request with "HttpOpenRequest" and adds the leaked data in request headers with "HttpAddRequestHeaders" in Fig.7. (e) M1 posts the cookies to web server with "HttpSendRequest" in Fig.7.

According to the call sequence: "NtOpenFile, NtReadFile, InternetOpen, HttpOpenRequest, HttpAddRequestHeaders, HttpSendRequest", we can get the sub PPN model as shown in Fig.13.

In Fig.13, we find that M1 sends the content of some sensitive local files to a remote FTP server. The leaked files include the username and password file of different applications such as ICQ, TotalCommand, Cuteftp and SmartFTP.

The PL_{global} is $\{p_{\text{ctg}} =$ "File", $p_{\text{cont}} =$ "Account file", $p_{\text{proc}} =$ "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$ -$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", $p_{\text{severity}} =$ ("high effect", "medium stealthy")$\}$.

6.2.2 Application-Associated Data Source

M2 is a variant of Netdex family. This malware is an exploiter. It exploits a security breach of Internet Explorer and creates and runs backdoor components. The call sequence fragments are shown in Fig.14.

According to the call sequence: "InternetGetCookie, InternetOpenW, FtpCreateDirectory, FtpPutFile", we can establish the sub PPN model, as shown in Fig.15.

①http://vxheaven.org/, Nov. 2015.

②http://download.cnet.com/, June 2015.

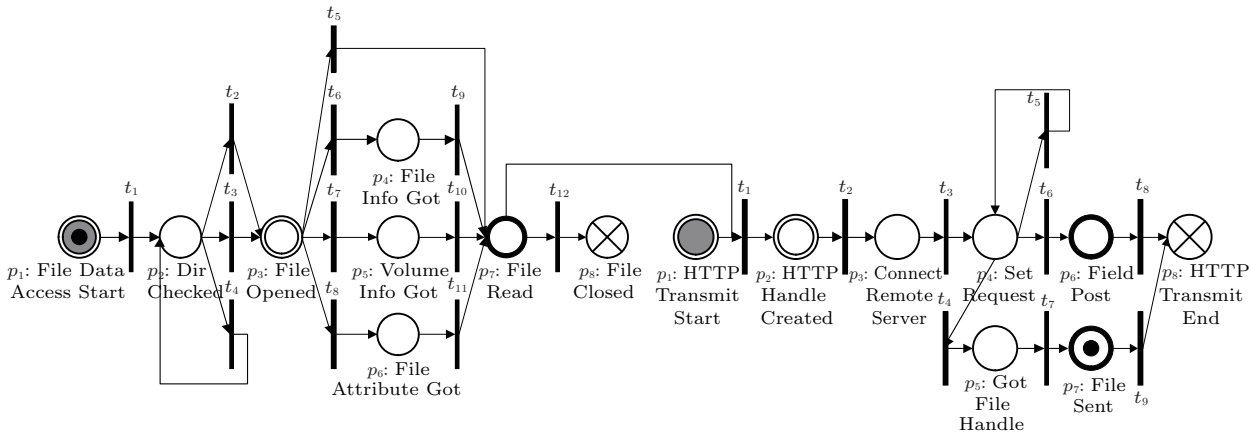


Fig.13. PPN for M1's behavior.

Fig.14. Call sequence fragment of M2. (a) M2 gets the cookies of selected website with "InternetGetCookie" in Fig.3. (b) M2 creates the Internet handle with "InternetOpen" in Fig.8. (c) M2 creates remote directory with "FtpCreateDirectory" and sends the local file to remote server with "FtpPutFile" in Fig.8.

In Fig.15, we find that the cookies of some websites are posted by M2 in the background to a remote server in accordance with FTP protocol.

The PL_{global} is {p_{ctg} = "App", p_{cont} = "Cookies", p_{proc} = "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$-$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", p_{severity} = ("medium effect", "low stealthy")}.

6.2.3 *System-Associated Data Source*

M3 is a variant of scanner family. This malware is a hacktool. It hides itself and leaks the system configuration and user info in order to help other malware. The call sequence fragments are listed in Fig.16.

According to the call sequence: "GetComputerName, GetUserName, Socket, Bind, Connect, Send", we can get the sub PPN model as shown in Fig.17. M3 collects the machine name and the user name, and sends them to its server by the background socket connection.

The PL_{global} is {p_{ctg} ="System", p_{cont} ="computer name" | "user name" | "network params", p_{proc} = "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$-$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", p_{severity} = ("low effect", "medium stealthy")}.

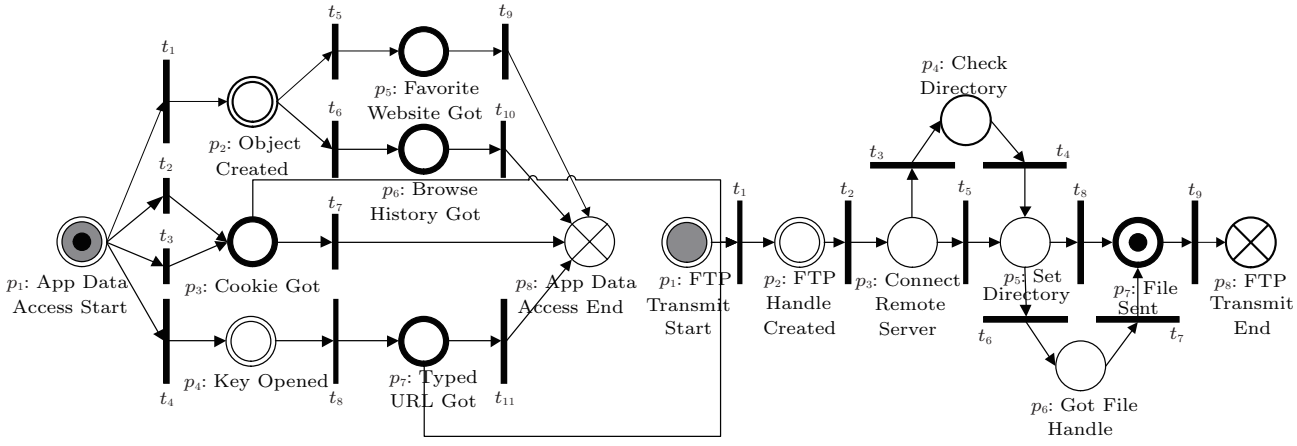


Fig.15. PPN for M2's behavior.

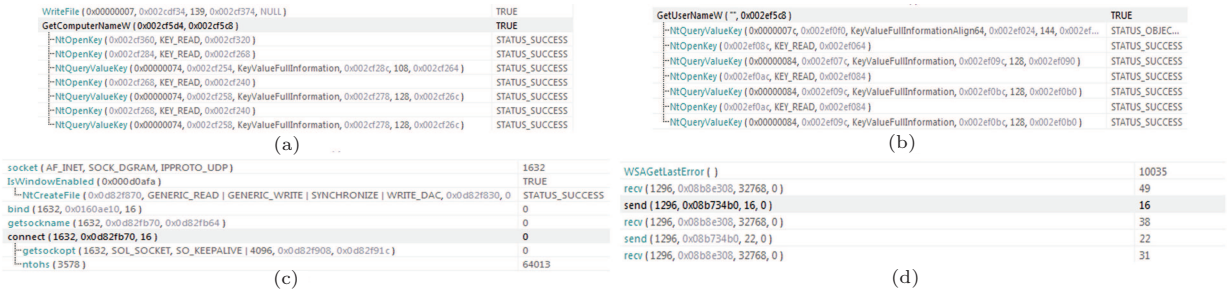


Fig.16. Call sequence fragment of M3. (a) M3 gets the machine name with "GetComputerName" in Fig.4. (b) M3 gets the user name with "GetUserName" in Fig.4. (c) M3 builds socket handle with "Socket" and "Bind", and then connects to remote server with "Connect" in Fig.6. (d) M3 sends the leaked data to remote server with "Send" in Fig.6.

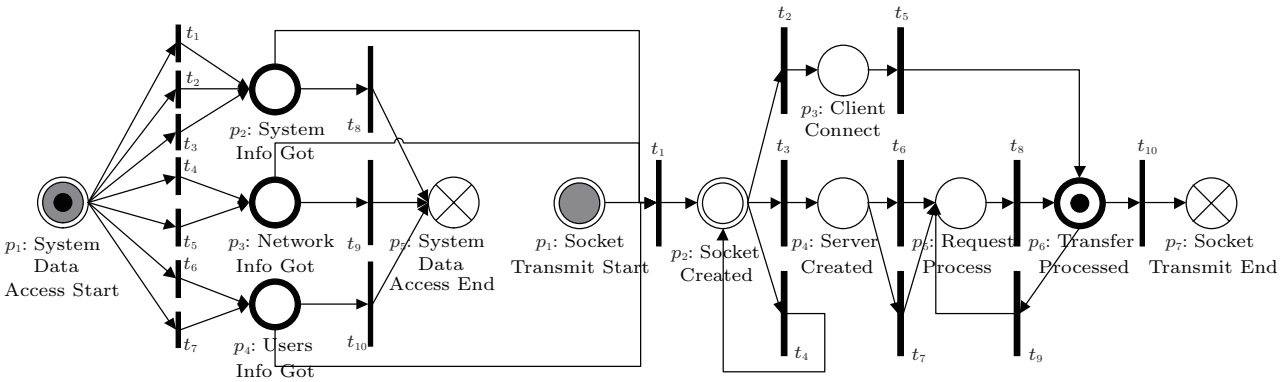


Fig.17. PPN for M3's behavior.

6.2.4 *Dynamic Data Source*

M4 is a variant of Aohy family, which is considered as a trojan. It hooks keyboard events and mouse events. The call sequence fragments are listed in Fig.18.

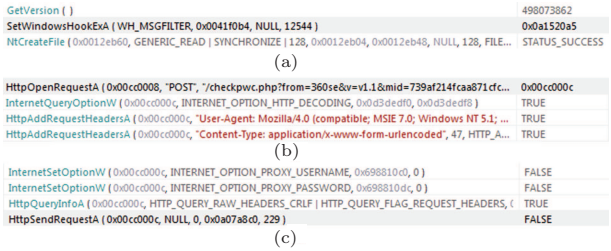


Fig.18. Call sequence fragment of M4. (a) M4 sets hook function with "SetWindowsHook" in Fig.5. (b) M4 builds socket handle with "Socket" and "Bind", and then connects to remote server with "Connect" in Fig.7. (c) M4 sends the leaked data to remote server with "Send" in Fig.7.

According to the call sequence: "SetWindowsHook, InternetOpen, InternetConnect, HttpOpenRequest, HttpSendRequest", we can get the sub PPN model, as shown in Fig.19. M4 leaks the keystoke log and sends to its server by the background HTTP connection.

The PL_{global} is {p_{ctg} = "Dynamic data", p_{cont} = "keystroke", p_{proc} = "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$-$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", p_{severity} = ("high effect", "high stealthy")}.

6.2.5 *Result Overview*

Based on the description of anti-virus software, we choose 133 malware samples which at least contain one kind of data privacy leak behaviors in the four kinds of privacy data source we classified and modeled above. We test the sample set with our approach and summarize the result in Table 8. We calculate the number of malware samples in each privacy data type, and every malware type is detected by our approach based on PPN. For instance, 19/20 in the Trojan/File Data cell means that we detect 19 out of 20 trojan samples which have privacy file data leak behavior.

As shown in Table 8, we have achieved 95% (121/128) detection rate for the whole sample set. Our approach effectively detects the behavior of privacy leaks of malware for various malware categories including trojan, backdoor, worm, hacktool, etc. We also prove that our approach can deal with different kinds of privacy data source such as file data, app data, sys-

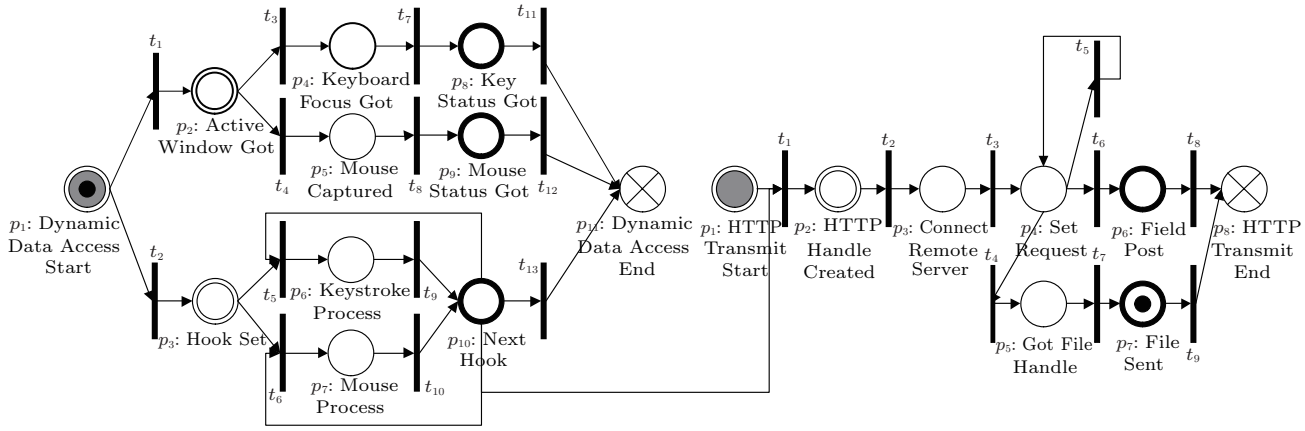


Fig.19. PPN for M4's behavior.

Table 8. Privacy Leak Behaviors Detection Result

	File Data	App Data	System Data	Dynamic Data	Sum
Trojan	19/20	5/5	4/4	49/52	**77/81**
Backdoor	1/1	12/13	5/6		**18/20**
Hacktool			9/10	5/5	**14/15**
Worm	3/3	1/1	1/1		**5/5**
Others	2/2	3/3	1/1	1/1	**7/7**
Sum	**25/26**	**21/22**	**20/22**	**57/58**	**121/128**

tem data and dynamic data.

According to the experimental results, we found useful information in three aspects.

1) Trojan samples with privacy leak behaviors are the majority of the malware categories (81/128, or 81 trojans in all the 128 samples) because of their unique characteristics. Trojan is designed to get control of target host from inside and then collect useful information, most of which concerns user privacy and has economic value. Specifically speaking, the dedicated trojan that leaks the bank account and online game accounts has formed a main branch in trojan family tree. Therefore, the privacy leak behaviors of trojans have been one of the most important factors to be considered in contemporary malware.

2) Backdoor malware tends to leak the application association privacy (13/20) and the system association privacy (6/20). Since backdoors usually utilize the vulnerability of applications or systems to set up covert entrance on target host, the related info is leaked for further movements and tactics of the attacker. Hacktool is similar to backdoor, which leaks system association privacy (10/15) even worse than backdoor. Worm in some occasions leaks privacy for fast and wide spreading, for instance, email-worm leaks the contact list for forwarding its copy.

3) In all the four privacy data sources, dynamic data leaks are of the most importance (58/128) because trojans (52/81) collect the privacy data such as account and password by recording user keystrokes. File data privacy leaks also outnumber other data sources because malware often searches for files storing privacy data such as configuration files storing user account name and password.

6.3 Suspicious Applications Case Study

In a similar way, we first choose one typical suspicious application sample for each kind of data source type. Because suspicious applications rarely collect the dynamic data source in our selected application set, we only list the samples of other three data sources.

6.3.1 Normal File Data Source

A1 is a p2p client. It is widely used for downloading and sharing file. The call sequence fragments are listed in Fig.20.

According to the call sequence: "NtOpenFile, NtReadFile, InternetOpen, FtpCreateDirectory, FtpPutFile", we can get the sub PPN model, as shown in Fig.21.

A1 sends a sensitive local file such as user contact info or favorite folder to a remote FTP server.

The PL_{global} is {p_{ctg} = "File", p_{cont} = "Contact",

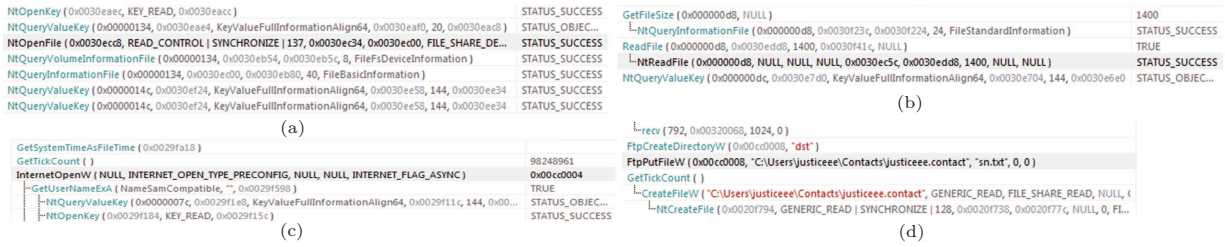


Fig. 20. Call sequence fragment of A1. (a) A1 creates a file handle with "NtOpenFile" in Fig.2. (b) A1 reads the local file with "NtReadFile" in Fig.2. (c) A1 creates an Internet handle with "InternetOpen" in Fig.7. (d) A1 creates remote directory with "FtpCreateDirectory" and sends the local file to remote server with "FtpPutFile" in Fig.7.

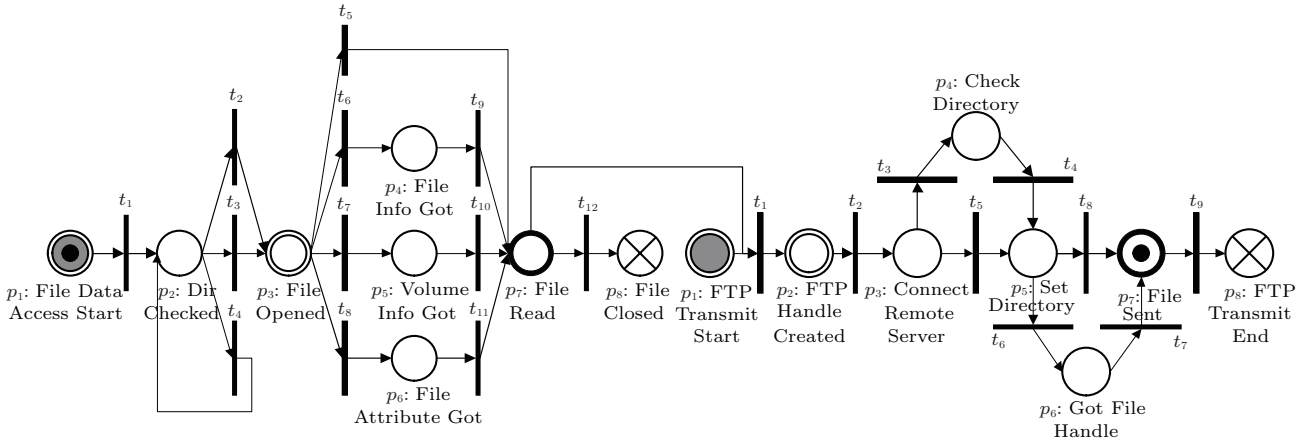


Fig.21. PPN for A1's behavior.

p_{proc} = "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$-$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", p_{severity} = ("medium effect", "medium stealthy")}.

6.3.2 *Application Associated Data Source*

A2 is a web browser, which has lots of users for its plug-in functionality. However, if a user installs a plug-in for managing the browse history and the favorite folder, it exposes leak behaviors. The call sequence fragments are listed in Fig.22.

According to the call sequence: "InternetGetCookie, InternetOpenW, HttpOpenRequest, HttpAddRequestHeader, HttpSendRequest", we can get the sub PPN model as in Fig.23.

The cookies of some website will be posted by A2 in the background to remote server with HTTP protocol.

The PL_{global} is {p_{ctg} = "App", p_{cont} = "Cookies", p_{proc} = "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$-$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", p_{severity} = ("medium effect", "low stealthy")}.

6.3.3 *System Associated Data Source*

A3 is a multimedia player. Its functionalities include online searching and media playing for new music or video. Fig.24 shows the call sequence fragments.

According to the call sequence: "GetComputerName, GetUserName, Socket, Bind, Connect, Send", we can get the sub PPN model as depicted in Fig.25.

Fig.22. Call sequence fragment of A2. (a) A2 gets the cookies of selected website with "InternetGetCookie" in Fig.3. (b) A2 creates the Internet handle with "InternetOpen" in Fig.6. (c) A2 creates Http request with "HttpOpenRequest" and adds the leak data in request headers with "HttpAddRequestHeaders" in Fig.6. (d) A2 posts the cookies to web server with "HttpSendRequest" in Fig.6.

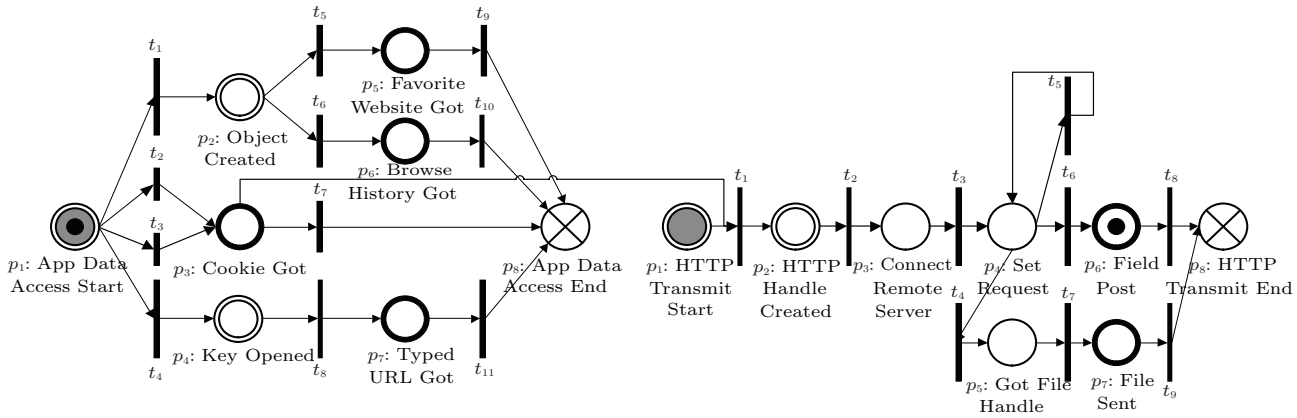


Fig.23. PPN for A2's behavior.

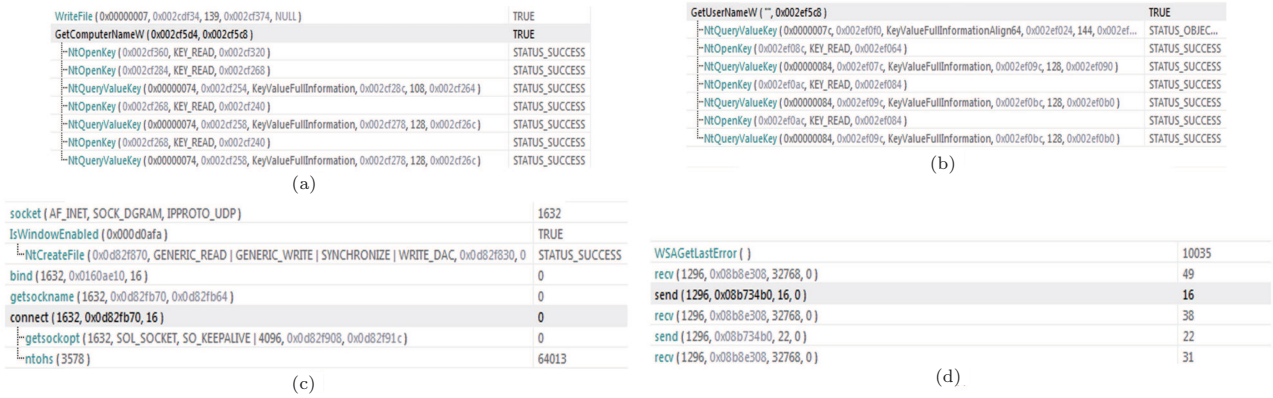


Fig. 24. Call sequence fragment of A3. (a) A3 gets the machine name with "GetComputerName" in Fig.4. (b) A3 gets the user name with "GetUserName" in Fig.4. (c) A3 builds socket handle with "Socket" and "Bind", and then connects to remote server with "Connect" in Fig.5. (d) A3 sends the leak data to remote server with "Send" in Fig.5.

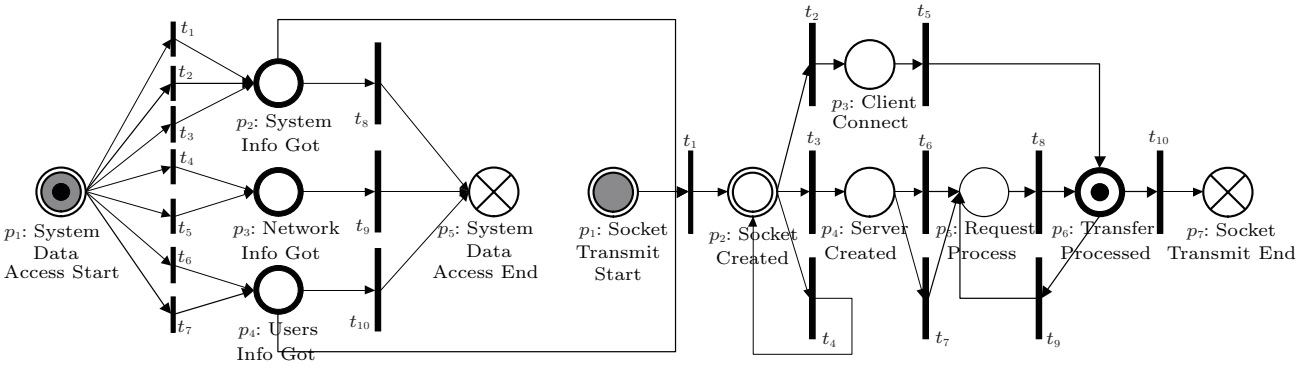


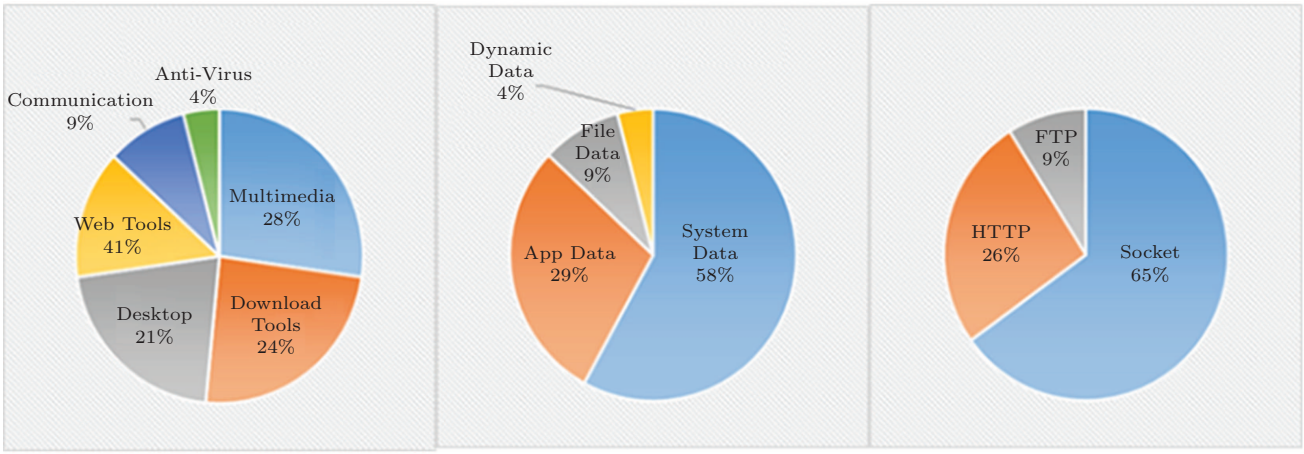
Fig.25. PPN for A3's behavior.

A3 collects the machine name and the user name and sends them to its server by the background socket connection.

The PL_{global} is {p_{ctg} = "System", p_{cont} = "computer name" | "user name" | "network params", p_{proc} = "$p_1t_1p_2t_4p_2t_3p_3t_7p_4t_9p_7$-$t_1p_2t_2p_3t_3p_4t_4p_5t_7p_7$", p_{severity} = ("low effect", "medium stealthy")}.

6.3.4 Result Overview

Although most of the popular applications are normal and are not going to harm the data privacy, there are still some suspicious samples that contain the privacy leak behaviors that we have defined, as shown in Fig.26. We summarize the experiment results and find



(a) (b) (c)

Fig.26. Overview of (a) content, (b) data source category and (c) connection type.

some useful information in three aspects as follows.

1) The desktop, download and multimedia applications contain more leak behaviors than other categories. We think that plug-in and bundles embedded in these applications are the main reason. Advertising companies or other third-party companies depend on these popular applications for business motives such as investigating user habits.

2) In different data sources, the system data are much more prone to leak than other kinds of data, such as the application data and the normal file data. This is because system information is not always defined as privacy. In fact, they are important security related data. As we mentioned in Section 2, system information such as user name, machine name can be used to leak the user's identity. The network parameters such as IP address and Mac address can also be used to leak the geographic information.

3) Most of the privacy leak applications use socket connection because it is easy to use and hard to detect. HTTP connection is widely used in web tools especially the web browsers since the application must build HTTP connection in the first place to complete its legal tasks. FTP connection is often used when the privacy data is stored in large or encrypted files.

6.4 Comparison and Discussion

To evaluate our PLS analysis approach based on PPN, we first compare it with recent related research work, and then consider commercial privacy related security software. Because privacy leak behavior is defined as a four-tuple: content, source, procedure and severity in this paper, we mainly evaluate the four features of different methods. Moreover, we discuss the extra aspects of privacy leak that the other methods consider to some extent.

6.4.1 PPN and Related Research Work

We compare three recent research studies with our method. Their names are Privacy Oracle[3], PiOS[4] and TaintDroid[5], which focus on privacy leak analysis. Privacy Oracle treats each application as a black box and reports on application leaks of user information via the network traffic that they send. PiOS uses static analysis to detect data flows in binary files to analyze programs for possible leaks of sensitive information from a mobile device to third parties. TaintDroid provides system-wide dynamic taint tracking and analysis for simultaneously tracking multiple sources of sensitive data. The comparison result is listed in Table 9. PPN can get content of leaked privacy data of all the four data sources, and record the runtime leak procedure. Privacy Oracle does not consider the privacy file data, and cannot trace the leak procedure as a black-box method. PiOS depends on static analysis so that it cannot get runtime content and procedure and trace the dynamic data source. TaintDroid does not take account of the application data source. PPN also evaluates the severity of the leak behavior, but none of the other three methods does.

6.4.2 PPN and Commercial Software

The privacy related commercial software considers privacy leak problems mainly from privacy data clearing, behavior monitoring and data protection. Accord-

Table 9. Compared with Other Research Work

	PPN	Privacy Oracle	PiOS	Taint Droid
Content	Runtime, any format	Runtime, limited format	Not runtime, any format	Runtime, any format
Source	All four sources	App system dynamic	App system	File system dynamic
Procedure	Runtime, full path	N/A	Not runtime, full path	Runtime, partial path
Severity	Effect, stealth	N/A	N/A	N/A

Table 10. Compared with Commercial Privacy Software

	PPN	Privacy Guardian	SpyAgent	Sophos Data Protection
Content	Runtime, any format	Not runtime, any format	Runtime, limited format	Runtime, any format
Source	All four sources	File app system	File dynamic	File system dynamic
Procedure	Runtime, full path	N/A	Runtime, partial path	N/A
Severity	Effect, stealth	Effect	Effect, stealth	N/A

ing to the description on ToptenReview③, we select Privacy Guardian, SpyAgent and Sophos Data Protection to make a comparison.

As shown in Table 10, the Privacy Guardian can scan and clear the privacy data which is stored as a log file, a config file or other usage trace file but cannot detect the privacy leak procedure of PLS. The SpyAgent can monitor many kinds of leak behavior such as file transferring and keylogger, but does not consider some important system data as privacy data. The Sophos Data Protection can filter unauthorized access to privacy data and block the outgoing connection to unknown remote server, only for selected files and well formatted data such as social security numbers and banking information.

7 Related Work

In this section, we first discuss the closest research findings to this paper which use Petri net as abstract model. We then present some work using other graphic and non-graphic models for software behavior modeling, especially the malicious behavior modeling. At last, some studies for software behavior analysis on other platforms such as smart phone are described.

Petri Net. To the best of our knowledge, we are the first ones to analyze the behavior of privacy leaks with Petri net[27-28]. But there are still lots of studies of analyzing other behaviors of applications or system with Petri net. Wang *et al.* for the first time presented Stochastic Game Nets (SGN) model and applied it in the competitive game analysis of network behavior[29-30]. They further solved the modeling and quantitative analysis of competitive game behaviors based on SGN[31-32]. Gao *et al.*[33] judged Trojan-like features of software using stochastic Petri nets and supported quantitative analysis for the behaviors of the target software. Tokhtabayev *et al.*[34] tried to find inter-process and intra-process malicious functionalities in software behaviors. The interesting functionalities were defined in the abstract system domain through activity diagrams and the specified functionalities were recognized by colored Petri net (CPN). They also built behavior based intrusion detection systems based on this approach to offer an effective solution against modern malware[35]. Liu *et al.*[36] used a combination of techniques from the behavior monitors and colored Petri net for detecting virus and worms. The malicious behavior was represented as Petri net and the notions of initial states and final states are used to define matching in this model. Ho *et al.*[37] proposed an intrusion detection architecture combining partial order planning and executable Petri nets to detect intrusions with multiple sources and intrusions where only incomplete behavioral data is available. They presented Partial Order State Transition Analysis to increase the flexibility of the traditional state analysis approach by allowing unordered events in the signature action sequence.

Other Graphic Models. Petri net is not the only graphic model that suits for analyzing behavior data, and other graphic models such as control flow graph, behavior graph and hierarchical behavior graph are also adopted to analyze the program behavior. Bruschi *et al.*[21] found that next generation malware will be characterized by the intense use of polymorphic and metamorphic techniques. They proposed a strategy for the detection of metamorphic malicious code inside a program P based on the comparison of the control flow

③http://www.toptenreviews.com/, June 2015.

graphs of P against the set of control flow graphs of known malware. Christodorescu *et al.*[22] defined a new graph representation of program behavior and a mining algorithm that constructs a malicious specification. Their algorithm inferred the system-call graphs from execution traces and derived a specification by computing the minimal differences between the system-call graphs of malicious and benign programs. Fredrikson *et al.*[24] implemented HOLMES to extract data dependence graphs and distinguish the malware from benign applications based on graph mining and concept analysis techniques. Martignoni *et al.*[23] addressed the semantic gap problem in behavioral monitoring by using hierarchical behavior graphs to infer high-level behaviors from myriad low-level events. Johnson *et al.*[38] proposed a differential slicing approach that automates the analysis of two runs of the same program which exhibits a difference in program state or output. A causal difference graph that captures the input differences that triggered the observed difference is outputted. Our approach may be similar to the control/data flow tracking about the privacy data tracing and checking, but our starting point is to describe software behavior with status transformation but not just to track the flow. Therefore, our approach has two main contributions. One is that ours can find more behavior detail besides data/control flow. The other one is that ours is flexible and can be easily expanded to be suitable for software behavior which is unrelated to data/control flow.

Other Formal Models. Some formal models without graphic presentation were also used to abstract the application behaviors from system calls. Christodorescu *et al.*[20] described malicious behavior by templates and presented a malware detection algorithm that addresses this deficiency by incorporating instruction semantics to detect malicious program traits. Jacob *et al.*[39] defined a generic approach for behavioral detection based on two layers. The abstraction layer is specific to a platform and a language. It interprets the collected instructions, API calls and arguments and classifies these operations. The detection layer relies on parallel automata parsing attribute-grammars where semantic rules are used for object typing (object classification) and object binding (data-flow). Kinder *et al.*[9] introduced the specification language CTPL (Computation Tree Predicate Logic) which extends the well-known logic CTL, and described an efficient model checking algorithm. Lanzi *et al.*[40] proposed a system-centric view to model the activity of benign programs. They argued that benign programs in general follow certain ways in which they use OS resources (such as the file system, the registry).

Other Platform. Local host is not the only victim of privacy leak behaviors. The wireless network and mobile device with growing performance and functionality has become another important platform on which lots of private data are stored. Lou and Ren[41] presented a novel authentication framework that integrates a new key management scheme to simultaneously achieve security, privacy, and accountability in wireless access networks. Liu *et al.*[42] proposed a new metric to quantify the system's resilience to location privacy attacks, and suggested using multiple mix zones to tackle this problem. A mathematical model is presented to treat the deployment of multiple mix zones as a cost constrained optimization problem. Lin *et al.*[43] proposed an efficient social-tier-assisted packet forwarding protocol, called STAP, for achieving receiver-location privacy preservation in VANETs. On smart phones, the detection and the analyzing of privacy leaks were discussed in recent studies. Egele *et al.*[4] studied the privacy threats of applications written for Apple's iOS, and presented a novel approach and a tool, PiOS, which uses static analysis to detect data flows and allow users to analyze programs for possible leaks of sensitive information from a mobile device to third parties. Enck *et al.*[5] provided the users with adequate control over and visibility into how third-party applications use their private data with TaintDroid, a system-wide dynamic taint tracking and analysis system capable of simultaneously tracking multiple sources of sensitive data. Gilbert *et al.*[44] proposed AppInspector, an automated privacy validation system that analyzes apps on smart phone and generates reports of potential privacy risks. Enck *et al.*[45] also unmasked the complexity of Android security and noted some possible development pitfalls that occurred when they defined an application's security.

8 Conclusions

In this paper, we presented an abstract model called PPN for private information leaks analysis to incorporate application dynamic behavior with network traffic. We built PPN of different privacy leak data sources and network connection to analyze the detail of private information leak. We applied our approach on real-world applications and found the details of the category, content, data source and destination of privacy leaks. In future work, we are going to apply privacy leaks to more platforms such as Linux systems and smart phone OSs.

References

[1] Backes M, Kopf B, Rybalchenko A. Automatic discovery and quantification of information leaks. In *Proc. the 30th IEEE Symposium on Security and Privacy*, May 2009, pp.141-153.

[2] Borders K, Prakash A. Quantifying information leaks in outbound Web traffic. In *Proc. the 30th IEEE Symposium on Security and Privacy*, May 2009, pp.129-140.

[3] Jung J, Sheth A, Greenstein B, Wetherall D, Maganis G, Kohno T. Privacy oracle: A system for finding application leaks with black box differential testing. In *Proc. the 15th ACM Conference on Computer and Communications Security*, Oct. 2008, pp.279-288.

[4] Egele M, Kruegel C, Kirda E, Vigna G. PiOS: Detecting privacy leaks in IOS applications. In *Proc. the 18th Annual Network & Distributed System Security Symposium*, Feb. 2011.

[5] Enck W, Gilbert P, Chun B G, Cox L P, Jung J, McDaniel P, Sheth A. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proc. the 9th USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2010, pp.393-407.

[6] Kirda E, Kruegel C. Behavior-based spyware detection. In *Proc. the 15th USENIX Security Symposium*, July 31-August 4, 2006.

[7] Egele M, Kruegel C, Kirda E, Yin H, Song D. Dynamic spyware analysis. In *Proc. the 2007 USENIX Annual Technical Conference*, June 2007, pp.233-246.

[8] Kruegel C, Kirda E, Mutz D, Robertson W, Vigna G. Polymorphic worm detection using structural information of executables. In *Proc. the 8th International Symposium on Recent Advances in Intrusion Detection*, Sept. 2005, pp.207-226.

[9] Kinder J, Katzenbeisser S, Schallhart C, Veith H. Detecting malicious code by model checking. In *Proc. the 2nd International Conference on Detection of Intrusions and Malware and Vulnerability Assessment*, July 2005, pp.174-187.

[10] Kruegel C, Robertson W, Vigna G. Detecting kernel-level rootkits through binary analysis. In *Proc. the 20th Annual Computer Security Applications Conference*, Dec. 2004, pp.91-100.

[11] Christodorescu M, Jha S. Static analysis of executables to detect malicious patterns. In *Proc. the 12th USENIX Security Symposium*, Aug. 2003.

[12] Moser A, Kruegel C, Kirda E. Limits of static analysis for malware detection. In *Proc. the 23rd Annual Computer Security Applications Conference*, Dec. 2007, pp.421-430.

[13] Sharif M, Lanzi A, Giffin J, Lee W. Impeding malware analysis using conditional code obfuscation. In *Proc. the 15th Annual Network and Distributed System Security Symposium*, Feb. 2008.

[14] Sharif M, Lanzi A, Giffin J, Lee W. Automatic reverse engineering of malware emulators. In *Proc. the 30th IEEE Symposium on Security and Privacy*, May 2009, pp.94-109.

[15] Rhee J, Riley R, Xu D, Jiang X. Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory. In *Proc. the 13th International Symposium on Recent Advances in Intrusion Detection*, Sept. 2010, pp.178-197.

[16] Lanzi A, Sharif M, Lee W. K-tracer: A system for extracting kernel malware behavior. In *Proc. the 16th Annual Network & Distributed System Security Symposium*, Feb. 2009.

[17] Yin H, Liang Z, Song D. HookFinder: Identifying and understanding malware hooking behaviors. In *Proc. the 15th Annual Network & Distributed System Security Symposium*, Feb. 2008.

[18] Moser A, Kruegel C, Kirda E. Exploring multiple execution paths for malware analysis. In *Proc. the 28th IEEE Symposium on Security and Privacy*, May 2007, pp.231-245.

[19] Comparetti P M, Salvaneschi G, Kirda E, Kolbitsch C, Kruegel C, Zanero S. Identifying dormant functionality in malware programs. In *Proc. the 31st IEEE Symposium on Security and Privacy*, May 2010, pp.61-76.

[20] Christodorescu M, Jha S, Seshia S A, Song D, Bryant R E. Semantics-aware malware detection. In *Proc. the 26th IEEE Symposium on Security and Privacy*, May 2005, pp.32-46.

[21] Bruschi D, Martignoni L, Monga M. Detecting self-mutating malware using control-flow graph matching. In *Proc. the 3rd International Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, July 2006, pp.129-143.

[22] Christodorescu M, Jha S, Kruegel C. Mining specifications of malicious behavior. In *Proc. the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Sept. 2007, pp.5-14.

[23] Martignoni L, Stinson E, Fredrikson M, Jha S, Mitchell J. A layered architecture for detecting malicious behaviors. In *Proc. the 11th International Symposium on Recent Advances in Intrusion Detection*, Sept. 2008, pp.78-97.

[24] Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X. Synthesizing near-optimal malware specifications from suspicious behaviors. In *Proc. the 31st IEEE Symposium on Security and Privacy*, May 2010, pp.45-60.

[25] Wang Y, Lin C, Ungsunan P D, Huang X. Modeling and survivability analysis of service composition using Stochastic Petri Nets. *The Journal of Supercomputing*, 2011, 56(1): 79-105.

[26] Yu M, Wang Y, Liu L, Cheng X. Modeling and analysis of email worm propagation based on stochastic game nets. In *Proc. the 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, Oct. 2011, pp.381-386.

[27] Fan L, Wang Y, Jin X, Li J, Cheng X, Jin S. Comprehensive quantitative analysis on privacy leak behavior. *PloS One*, 2013, 8(9): e73410.

[28] Fan L, Wang Y, Cheng X, Li J, Jin S. Privacy theft malware multi-process collaboration analysis. *Security and Communication Networks*, 2015, 8(1): 51-67.

[29] Wang Y, Lin C, Meng K, Yang H, Lv J. Security analysis for online banking system using hierarchical stochastic game nets model. In *Proc. IEEE Global Communications Conference*, Nov. 30-Dec. 4, 2009.

[30] Wang Y, Lin C, Wang Y, Meng K. Security analysis of enterprise network based on stochastic game nets model. In *Proc. IEEE International Conference on Communications*, June 2009.

[31] Wang Y, Lin C, Meng K, Lv J. Analysis of attack actions for e-commerce based on stochastic game nets model. *Journal of Computers*, 2009, 4(6): 461-468.

[32] Wang Y, Yu M, Li J, Meng K, Lin C, Cheng X. Stochastic game net and applications in security analysis for enterprise network. *International Journal of Information Security*, 2012, 11(1): 41-52.

[33] Gao H, Wang Y, Wang L, Liu L, Li J, Cheng X. Trojan characteristics analysis based on Stochastic Petri Nets. In *Proc. IEEE International Conference on Intelligence and Security Informatics*, July 2011, pp.213-215.

[34] Tokhtabayev A, Skormin V, Dolgikh A. Dynamic, resilient detection of complex malicious functionalities in the system call domain. In *Proc. Military Communications Conference*, Oct. 31-Nov. 3, 2010, pp.1349-1356.

[35] Tokhtabayev A, Skormin V, Dolgikh A. Expressive, efficient and obfuscation resilient behavior based IDs. In *Proc. the 15th European Symposium on Research in Computer Security*, Sept. 2010, pp.698-715.

[36] Liu P, Wang J, He D. Worm detection using CPN. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2004, pp.4941-4946.

[37] Ho Y, Frincke D, Tobin D. Planning, Petri nets, and intrusion detection. In *Proc. the 21st National Information Systems Security Conference*, Oct. 1998.

[38] Johnson N M, Caballero J, Chen K Z, McCamant S, Poosankam P, Reynaud D, Song D. Differential slicing: Identifying causal execution differences for security applications. In *Proc. the 32nd IEEE Symposium on Security and Privacy*, May 2011, pp.347-362.

[39] Jacob G, Debar H, Filiol E. Malware behavioral detection by attribute-automata using abstraction from platform and language. In *Proc. the 12th International Symposium on Recent Advances in Intrusion Detection*, Sept. 2009, pp.81-100.

[40] Lanzi A, Balzarotti D, Kruegel C, Christodorescu M, Kirda E. AccessMiner: Using system-centric models for malware protection. In *Proc. the 17th ACM Conference on Computer and Communications Security*, Oct. 2010, pp.399-412.

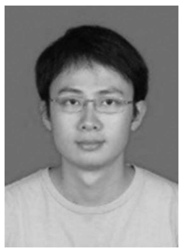
[41] Lou W, Ren K. Security, privacy, and accountability in wireless access networks. *IEEE Wireless Communications*, 2009, 16(4): 80-87.

[42] Liu X, Zhao H, Pan M, Yue H, Li X, Fang Y. Traffic-aware multiple mix zone placement for protecting location privacy. In *Proc. INFOCOM*, Mar. 2012, pp.972-980.

[43] Lin X, Lu R, Liang X, Shen X. STAP: A social-tier-assisted packet forwarding protocol for achieving receiver-location privacy preservation in VANETs. In *Proc. INFOCOM*, Apr. 2011, pp.2147-2155.

[44] Gilbert P, Chun B G, Cox L P, Jung J. Automating privacy testing of smartphone applications. Technical Report, TR-CS-2011-02, Duke University, 2011.

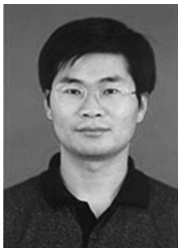
[45] Enck W, Ongtang M, McDaniel P. Understanding Android security. *IEEE Security & Privacy*, 2009, 7(1): 50-57.



Le-Jun Fan received his B.S. and M.S. degrees in circuit and system from the University of Science and Technology of China, Hefei, in 2004 and 2007 respectively. He received his Ph.D. degree in information security in 2013 from the Research Center of Web Data Science & Engineering of the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include software malicious behavior analysis, data privacy and Petri nets. He is a member of IEEE. He is now working at National Computer Network Emergency Response Technical Team/Coordination Center of China.

Yuan-Zhuo Wang is an associate professor at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His current research interests include network and information security analysis, Web behavior analysis, stochastic Petri nets and stochastic game nets. So far he has published over 90 publications in journals and international conferences. He is a senior member of CCF, ACM, and IEEE. He is currently serving on Transactions on Parallel and Distributed Systems, International Journal of Internet Protocol Technology, Journal of Parallel and Distributed Computing, Journal of Computer Science and Technology, Chinese Journal of Computers, and Chinese Journal of Electronics.

Jing-Yuan Li is with Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He got his Bachelor's degree in computer science at Harbin Institute of Technology in 2004, and his Ph.D. degree in computer software and theory at University of Science and Technology of China, Hefei, in 2009. His research interests includes stochastic game theory, data mining in social network and social media, etc.



Xue-Qi Cheng is a professor at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), and the director of the Research Center of Web Data Science & Engineering (WDSE) in ICT, CAS. His main research interests include network science, Web search and data mining, P2P and distributed system, and information security. He has published over 100 papers in prestigious journals and international conferences, including New Journal of Physics, Physical Review E, Journal of Statistical Mechanics, IEEE Trans. Information Theory, SIGIR, WWW, CIKM, WSDM, AAAI, IJCAI and so on. He is currently serving on the editorial board of Journal of Computer Science and Technology, Journal of Computer Research and Development, and Chinese Journal of Computers. Professor Cheng is a recipient of the Youth Science and Technology Award of Maoyisheng Science and Technology Award (2008), the CVIC Software Engineering Award (2008), the Excellent Teachers Award from Graduate University of Chinese Academy of Sciences (2006), the Second-Class Prize for the National Science and Technology Progress (2004), the Second-Class Prize for the Chinese Academy of Sciences and Technology Progress (2002), and the Young Scholar Fellowship of Chinese Academy Sciences (2000).

Chuang Lin is a professor of the Department of Computer Science and Technology, Tsinghua University, Beijing. He is an Honorary Visiting Professor of University of Bradford, UK. He received his Ph.D. degree in computer science from Tsinghua University in 1994. His current research interests include computer networks, performance evaluation, network security analysis, and Petri net theory and its applications. He has published more than 600 papers in research journals and IEEE conference proceedings in these areas and has published six books. Professor Lin serves as the technical program vice chair of the 10th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004), the general chair of ACM SIGCOMM Asia Workshop 2005, and the 2010 IEEE International Workshop on Quality of Service (IWQoS 2010). He is an associate editor of IEEE Transactions on Vehicular Technology and an area editor of Journal of Computer Networks.