International Journal of Grid and Distributed Computing
Vol. 6, No. 2, April, 2013

# SCMA: Scalable and Collaborative Malware Analysis using System Call Sequences

Huabiao Lu, Xiaofeng Wang and Jinshu Su
*School of Computer, National University of Defense Technology, Changsha, China*
ccmaxluna@gmail.com, {xf_wang, sjs}@nudt.edu.cn

**Abstract**

*Malware is huge and growing at an exponential pace. Symantec observes 403 million new malware samples in 2011. Therefore, that efficiently and effectively analysis so many malware samples becomes a great challenge. Centralized systems cause problems of single point of failure as well as processing bottlenecks. Previous distributed systems are mainly applied for specific or simple malware. This paper presents SCMA, a new distributed malware analysis system which can analyze various malware, shares behavior fragments among its monitors efficiently, analyzes malware based on global behavior of malware and aggregates those analyses among monitors in a load-balance way. We implemented a proof-of-concept version of SCMA and performed experiments with 917 real-world malware samples; preliminary results from our evaluation confirm that SCMA has comparable performance with centralized system, but much better scalability, and is approximately consistent with the analysis of AV scanners.*
**Keywords :** *collaborative malware analysis, System Call Sequences, local analysis, global aggregation, rendezvous-based sharing structure*

## 1 Introduction

Malware, short for malicious software, is software designed to disrupt computer operation, gather sensitive information, or gain unauthorized access to a computer system [1]. Malware includes computer viruses, worms, botnet, trojan horses, spyware, and so on. Malware is one of the most serious security threats on the Internet [2]. In fact, malware is the cause of most Internet problems such as spams and DoS (Denial of Service) [2]. As a result, malware are the most serious threats to Internet.

The battle against malware is becoming more difficult. The volume of new malware, fueled by easy-to-use malware morphing engines, is growing at an exponential pace [3]. Derived from Symantec Internet Security Threat Report (Volume 3-16), Figure 1 shows that the number of new malware samples observed by Symantec increases approximately at an exponential pace from 2002 to 2011. Symantec observes 403 million new malware samples in 2011 [3], average 1.1 million malware samples per day. At the same time, security organizations usually deploy numerous wide-spread sensors to monitor the comprehensive malware samples, e.g. . How to organize those large number of sensors and analysis such many malware samples is a challenge.

The battle against malware is becoming more difficult because of new challenges emerging. The new challenges are as follows. First, the number of new malware samples is huge

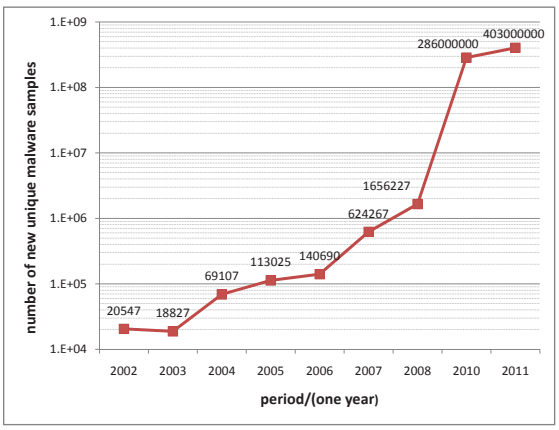



**Figure 1. Number of new malware samples each year observed by Symantec [1]**

and growing at an exponential pace. Derived from Symantec Internet Security Threat Report (Volume 3-16) [1], Figure 1 shows that the number of new malware samples observed by Symantec increases approximately at an exponential pace from 2002 to 2011. During 2006, Symantec observes only 140,690 unique by hash malware samples, while after 5 years, Symantec observes 403 million in 2011, average 1.1 million unique malware samples per day. Second, various kinds of malware are diversity and the infected hosts are evolving. As a result, reputation-based detection systems are ineffective, e.g. DNS blacklists only cover 17.18% of Conficker (Downadup) [2] victims on any of four famous DNS blacklists (DNS-BL, SORBS, Spamhaus, and SpamCop) [3], and, most heavily infected ASes by Conficker are not shown in the top 500 malicious ASes of FIRE [4] and only 0.33% hosts and 0.2% of ASes are reported by Dshield [5] as malicious. Third, malware authors utilize various sophisticated technologies to evade detection, e.g. code obfuscation, payload encryption, random rendezvous domains, and multiple attack vectors (e.g. vulnerability, File sharing, removable devices) [6]. Fourth, samples of the same family spread throughout the whole Internet and behave stealthy, thus, from a single area, such as a single enterprise, a single AS, we can see only very small part of the whole samples of a family. Because of sophisticated technologies used, samples behave similarly but very differently with other samples of the same family. Thus, to gain the intrinsic behaviors of a family, we need collect enough samples of the family and find the invariant behaviors in those various samples. However, it is difficult to discover and identify samples which are stealthy, and meantime samples are widespread, therefore, we will collect very limited samples from a single area. Hence, to gain enough samples of a family, we need to discover and collect samples from multiple areas of Internet. Intrinsic behaviors of a family are used to detect new emerging variants of the family and block continued dissemination of the family. Those intrinsic behaviors are called the signature of the family.

Existing works on malware detection are clustered into two categories: (1) centralized systems which collect widespread suspicious programs to central servers where centralized algorithm determines whether suspicious programs are malicious or not [7-11]; (2) distributed system, consisting of multiple monitors, each monitor of which monitors an area, collects suspicious programs from the area and shares the attributes of suspicious programs among other monitors to collaboratively determine whether suspicious programs are malicious or not [12, 13, 29].

Centralized malware detection system monitors one or multiple areas, and collects all suspicious programs to a centralized server to determine which programs are malicious. If only monitoring one or a few areas, as malware samples of a family are widespread and stealthy, the system sees very limited samples of a family and has no sufficient proofs to valid the malicious characteristics of malware. Hence, this kind of centralized systems are either ineffective, or using some prior knowledge and not advancing with the change of malware. BotHunter [7] recognizes a correlated dialog trail consisting of multiple stages to detect host infected by bot. However, BotHunter mainly utilizes predefined rules instead of automated methods to recognize each stage. Thus, centralized system monitoring multiple areas is dominant. However, as the number of malware samples huge and growing at an exponential pace, this kind of centralized system will introduce processing bottleneck if it collects the mass traces of numerous suspicious programs to central severs [8, 9], or will loss valuable information if it only collects compact profiles of samples [10,11].

Distirbuted malware detection system has multiple monitors, each monitors an area and shares the attributes of suspicious programs to determine whether suspicious programs are malicious or not. This kind of malware detection system is scalable and accurate, has broad view, and can monitor the global Internet or all the smartphones. However, as various sophisticated technologies used, malware samples of a family monitored by different monitors exhibit similar but very different behaviors. Therefore, the main challenge in this kind systems is how to identify the distributed samples belonging to the same family. Existing works in this kind have not yet solved the main challenge, inducing that they are only applicable for some simple malware [12, 13], or only applicable for malware in specific stage[29]. Min et al. [12] presented a system that collaboratively detects mono-morphic worm [1], by fingerprinting mono-morphic worm as single strings of fixed length (the fixed length is enough long, such as 40 bytes) in packet and generates byte-signatures for them by filtering and aggregating single string fingerprints at distributed monitors in multiple edge networks. Autograph [13] shares IP addresses of port scanners among distributed monitors by expanding the view of scan to speed-up the worm detection speed in single monitors. Vigilante [29] detects worms by instrumenting vulnerable programs to analyze infection attempts at hosts, and generates SCAs (short for self-certifying alerts, security alerts that can be inexpensively verified by any vulnerable host autonomously) if infection attempt is detected and broadcasts SCAs over an overlay network to other hosts to contain an outbreak collaboratively. Those distributed systems are scalable, detecting malware very quickly, but are only applicable to mono-morphic worm [12], scan worm [13] or vulnerability-based worm at propagation stage.

In this paper, we focus on distributed malware detection methods. We propose an end-network cooperated, spatial correlated and collaborative malware detection system, SCMA, which can intelligently identify the family that the distributed samples belong to and anti various sophisticated technologies. There are several challenges: 1) how to discover suspicious programs at end devices where most programs are benign and malware samples may use sophisticated technologies, e.g. rootkit, code obfuscation, payload encryption, and multiple attack vectors; 2) how to identify which suspicious programs belongs to same family collaboratively while each monitor has only part of suspicious programs and suspicious programs of a family maybe behave very differently; 3) how to effectively determine which

[1]Worm is a kind of malware that replicates itself in order to spread to other computers, it often uses a computer network to spread itself [32]. Mono-morphic worm is a specific worm, which spread itself without any changes, thus, samples of a mono-morphic worm are looked the same, behave as same

family is malicious while there is no centralized point to gather enough samples of the family. SCMA utilizes three intrinsic characteristics of recent popular malwares to solve those challenges: (1) samples are automatic programs without requiring human-driven activities, and utilize network interaction to perpetrate their malicious activities, e.g., spamming, stealing private data, downloading malware updates, etc; (2) samples of the same family behave very differently but still have some similarity, and therefore, they have some same behavior fragments; (3) end devices infected by samples of a family spreads widely, hence, the dispersion of those infected end devices are large.

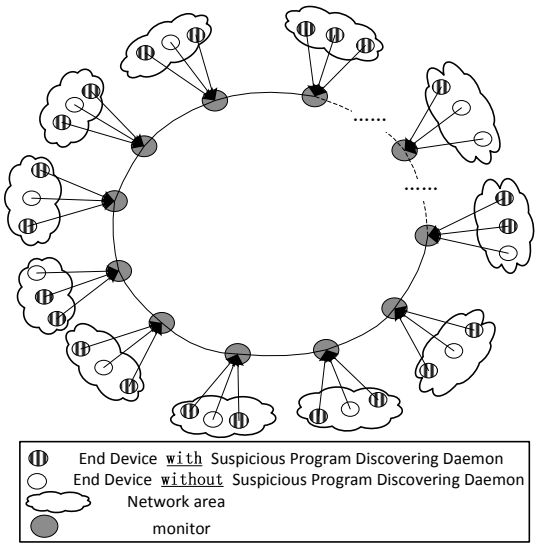
We take full advantage of these three intrinsic characteristics. Firstly, we identify the program that is automatic without human-driven activities and has network interaction as suspicious program; secondly, monitor shares behavior fragments of suspicious programs with other monitors in a style like peer-to-peer to gain the overall dispersion of each fragment; thirdly, monitor utilizes a heuristic correlation algorithm to derive the global dispersion of the family of suspicious program locally from the global dispersion of the fragments; Finally, monitor identifies the program as malicious locally if the global dispersion of its family is greater than a predefined threshold. All in all, the main contributions of this paper are as follows:

1. We design a lightweight suspicious program discovering daemon to find suspicious programs which are automatic and with network interaction. the daemon monitors keyboard/mouse and network interaction events for each programs in end devices by hooking system API. The daemon submits the behaviors of identified suspicious programs to a monitor that monitors the area of the end device. The daemon is lightweight increasing little interference to end device, effective, and can anti various sophisticated technologies.
2. We propose a distributed heuristic malware identify algorithm. It firstly recognizes the common behavior fragment set shared by the programs of the same family. It splits the behavior of suspicious program into fragments, shares those fragments with other monitors to gain the overall dispersion of each fragment, and heuristically identifies the fragment set which are rare in benign programs and maximizes the dispersion of the set as the common fragment set; Then it determines whether the suspicious program is malicious or not based on the dispersion of its family while the dispersion is induced from global dispersion of fragments in its common fragment set.
3. We introduce a novel RENdezvous-based Sharing structure (RENShare) to share behavior fragments between monitors efficiently and calculate the global dispersion of each fragment.

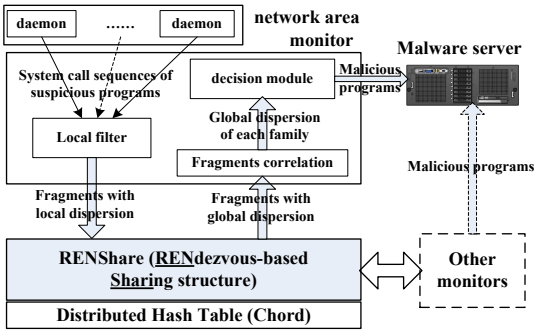
The rest of this paper is organized as follows.In section 2, we firstly describe the system overview of SCMA, and then present its main components in detail. We validate our method in a testbed using real malware samples in section 3. Finally, we conclude it and present future works in section 4.

## 2 System Overview

In this section we firstly describe the architecture of SCMA, and then describe its main components in detail.




(a) Deployment of SCMA




(b) structure of Monitor in SCMA

**Figure 2. Architecture of SCMA**

## 2.1 Architecture of SCMA

Figure 2 shows the architecture of our SCMA malware detection system. SCMA consists of three parts: (1) the lightweight suspicious program discovering daemon that deployed on end devices; and (2) the monitors that receive the behaviors (that is system call sequences in this paper) of suspicious programs from the daemons, split the system call sequences into fragments, share the fragments with other monitors to gain global dispersion of each fragment, utilizes a heuristic correlation algorithm to derive the global dispersion for each family from global dispersion of fragments, determine whether the program is malicious or not based on the global dispersion of its family, and finally, send the malicious programs to malware server for further analysis and verifying; (3) RENShare, a concept component used for sharing fragments among monitors, which receives local dispersion of fragments from all monitors and return global dispersion of fragments to source monitors that submit those fragments. Figure 2(a) describes the deployment of SCMA. SCMA consists of several monitors, each of which monitors a network area (e.g. LAN, enterprise network, an area of smartphones, AS), In each monitored network area, some of its end devices are deployed

with the suspicious program discovering daemon and others are not, but one monitor MUST be deployed in each network area. Figure 2(b) presents the detailed structure of the monitor in SCMA.

### 2.2 Suspicious Program Discovering Daemon

Since most programs in end device (including end-host, smartphone, and so on) are benign, it is relatively inefficient to gain system calls sequences for all programs. Thus, our suspicious program discovering daemon is designed to sift benign programs out before monitoring system calls sequences.

The daemon monitors keyboard/mouse events of the device to uncover which program has human activity/interaction. To do this, our daemon hooks OS (Operating System) system calls related to keyboard/mouse events, and also determines the program generating them. At this time, we also consider that whether the events are produced by real physical devices or not. Our daemon only trusts the events from physical devices.

The daemon employs three metrics: (1) time difference between the time when a process issues a network interaction and the prior time when a process produces a mouse/keyboard event, (2) the source of the events, and (3) whether a process is running foreground or not at the time. Daemon recognizes the program as benign if (1) the time difference is very small, (2) the event is from actual physical devices, and (3) the process is running foreground. Otherwise, identifies the program as suspicious after a predefined period.

After the program is identified as suspicious, daemon monitors its system call activities, and sends the system call sequences to the monitor responsible for the network area. Although daemon will loss the system calls that occur prior to the program being discovered as suspicious, logging the remaining calls is sufficient because most malwares do malicious behaviors not just one time.

Runtime verification aims to check whether an application executes its behavior as specified. Thereby, we can use the more complex technology– runtime verification [23]-to discover the suspicious programs by checking the above mentioned three conditions. Runtime verification will be our future work for suspicious programs discovering.

As for smartphone, suspicious programs discovering daemon will introduce unneglectable interference to end-users. Instead of deploying the daemon in a smartphone directly, we can emulate the smartphone at a region server as Paranoid Android [31] does, and then deploy the daemon at the emluator. Paranoid Android [31] emulate smartphone on virtual machine (VM) by transmitting the trace of smartphone to VM online and replaying the trace on VM.

### 2.3 LOGLOG Address Set Representation

The naive method to calculate the global address set of infected end devices $H_g(j)$ for a given fragment $j$, is performing the union operation on all local address sets $H_i(j)$ $(i = 1, 2, \cdots, n)$, a.k.a.,$H_g(j) = \sum_{i=1}^{n} H_i(j)$, where $H_i(j)$ presents the local address set of fragment $j$ in monitor $i$ and $n$ is the number of monitors in SCMA. However, transmitting the address set among monitors directly is a bandwidth dense task. And meanwhile, we only need to estimate the number of distinct IP addresses in address set, i.e. the cardinality of address set. Hence, we need to find some compact representation to reduce communication and computation overhead. Since the address sets may vary from empty to very large, we

would best use a very small size memory to estimate cardinalities varying in several orders of magnitude. We use the LOGLOG counting proposed by Durand and Flajolet [17] as the address set presentation.

The LOGLOG counting counts large cardinalities with high accuracy and little storage. Firstly, it uses a uniform hash function to transform elements of the set into sufficiently long binary strings (hashed value), in such a way that bits composing the hashed value closely resemble random uniform independent bits. Apparently, using uniform hash function can eliminate duplicates in the set. For a binary strings (hashed value) $x \in \{0,1\}^l$, let $\rho(x)$ denote the position of its first 1-bit, e.g. $\rho(1\cdots) = 1$, $\rho(001\cdots) = 3$. Secondly, it separates elements into m groups also called "buckets", where m is a design parameter. If $m = 2^k$, this is easily done by using the first k bits of $x$ as the index of a bucket. And the counter denoted by $M^{(u)}$ records the largest $\rho$ of hash values in bucket $u$. There are m counters: $M^{(u)}$ $(u = 1, 2, \cdots, m)$. Lastly, after $m$ counters recording all the elements of the set, the arithmetic mean $\frac{1}{m}\sum_{i=1}^{m} M^{(i)}$, can legitimately be expected to approximate $\log_2(\frac{N}{m})$ plus an additive bias, where $N$ is the cardinality of the set. The estimator of $N$ is $\tilde{N}$ presented as Formula 1.

$$\tilde{N} = \alpha_m m 2^{\frac{1}{m}\sum_{i=1}^{m} M^{(i)}} \tag{1}$$

where constant $\alpha_m = (\Gamma(-\frac{1}{m})\frac{1-2^{\frac{1}{m}}}{\log 2})^{-m}$, while $\Gamma(s) = \frac{1}{s}\int_0^\infty e^{-t}t^s dt$. In practical implementations, $\alpha_m \sim \alpha_\infty - \frac{2\pi^2+\log^2 2}{48m}$, while $\alpha_\infty = e^{(}-\gamma)\frac{\sqrt{2}}{2} \doteq 0.39701$ ($\gamma$ is Eulers constant), generally speaking, $\alpha_m$ can be replaced by $\alpha_\infty$.

The LOGLOG counting provides asymptotically a valid estimator of $N$. The standard error, which measures in a mean-quadratic sense and in proportion to $N$ the deviations to be expected, is closely approximated as the formula 2.

$$Standard \quad error \approx \frac{1.05}{\sqrt{m}} \tag{2}$$

As proved by Durand and Flajolet [17], each counter that records the largest $\rho$ of hash values in its bucket only need to count from 1 to $\lceil \log_2(\frac{N_{max}}{m}) + 3 \rceil$, hence, each counter only need $\lceil \log_2 \lceil \log_2(\frac{N_{max}}{m}) + 3 \rceil \rceil$ bits. $N_{max}$ is an a priori upperbound on cardinalities, thus, $N_{max}$ is $2^{32}$ for IPv4 address, and we only need to allocate 5 bits for each counter. Assuming $m = 1024$, the $m$ counters of LOGLOG counting all need 5Kbits and the standard error is 3.3%. The $m$ counters of LOGLOG counting is called LOGLOG address set representation in following sections.

### 2.4 Local Filter

**Definition 1: Behavior fragments with length q**. Giving a system calls sequence $C = (C_1, C_2, C_3, \cdots, C_c)$, the subsequences $(C_1, C_2, \cdots, C_q)$, $(C_2, C_3, \cdots, C_{q+1})$, $\cdots$, $(C_{c-q+2}, C_{c-q+1}, \cdots, C_{c-1})$, $(C_{c-q+1}, C_{c-q}, \cdots, C_c)$ are called **behavior fragments with length q**, or **fragments** for short.

Local filter module receives the system call sequences of suspicious programs from the daemons deployed in its network area, splits the system call sequences into fragments with a predefined length, selects representative fragments, calculates the local LOGLOG address set presentation for each representative fragment using **Algorithm 1** in which IP address set $IPset_i(j)$ for fragment $j$ presents the IP addresses of end devices which submit

a system call sequence containing the fragment $j$ to monitor $i$, then sends the fragment $j$ carrying with its local LOGLOG address set presentation $\mathbf{P_i(j)}$ to RENShare. Here, representative fragments are selected by a window sampling algorithm [18]. The window sampling algorithm selects the one with a minimal hash value within a given window as a representative one, then slides the window and selects the next one.

**Algorithm 1** calculate the local LOGLOG address set presentation

**Input:** IP address set $IPset_i(j)$ of fragment $j$ at monitor $i$, and the number of counters $m$
**Output:** The $m$ counters $M_i^{(u)}(j)$ $(u = 1, 2, \cdots, m)$ in $\mathbf{P_i(j)}$

1: allocate $m$ counters: $M_i^{(u)}(j)$ $(u = 1, 2, \cdots, m)$, each counter is 5 bits.
2: Initialize all $m$ counters to zero, $M_i^{(u)}(j) = 0$ $(u = 1, 2, \cdots, m)$.
3: **for** $IP_v \in IPset_i(j)$ **do**
4: generate the binary strings $Hash_v$ for $IP_v$ using a uniform hash function, $Hash_v = hash(IP_v)$.
5: calculate the first 1-bit position $\rho(v)$ for $Hash_v$.
6: calculate the counter index $u_v$ for $Hash_v$: $u_v = Hash_v \% m$ .
7: **if** $M_i^{(u_v)}(j) < \rho(v)$ **then**
8: $M_i^{(u_v)}(j) = \rho(v)$
9: **end if**
10: **end for**

### 2.5 RENShare (RENdezvous-based Sharing Structure)

RENdezvous-based Sharing structure (RENShare) is a DHT based information sharing infrastructure; unlike ordinary peer-to-peer network which only efficiently locate the node that responsible for the specific key, in addition, RENShare accumulates the global statistics of each key. In SCMA, RENShare receives a lot of fragments from all monitors, calculates the global LOGLOG address set presentation for each fragment by "max-merge" the local LOGLOG address set presentation of the same fragment submitted by different monitors, and then returns the global LOGLOG address set presentation to the source monitors which submit the fragment to RENShare. RENShare, a concept component, embeds itself among the monitors of SCMA.

Assuming that there are 16 ($n = 16$) monitors in SCMA: $s_0, s_1, s_2, \cdots, s_{15}$, Figure 3 illustrates the functionality of RENShare. Each monitor selects some representative behavior fragments, as Figure 3(a) describes, each monitor $s_i (0 \leq i \leq 15)$ has $k_i (0 \leq i \leq 15)$ distinct representative fragments, e.g. monitor $s_0$ has fragments: $frag_{s0\_1}$, $frag_{s0\_2}$, $frag_{s0\_3}$, $\cdots$, $frag_{s0\_k0}$. Firstly each monitor sends selected fragments to RENShare as traditional DHT network does, mapping each fragment to a key, and routes the fragment with its local LOGLOG address set presentation to a monitor that responsible for the key. As DHT infrastructure implied, the same fragment from differnet monitors will be routed to the same monitor, as Figure 3(b) describes, $frag_{s0\_2}$, $frag_{s3\_1}$, $frag_{s5\_3}$, $frag_{s8\_1}$, $frag_{s13\_3}$ are same fragments and converged to monitor $s_6$. But, unlike traditional DHT network, RENShare not just routes the fragment to a monitor that responsible for it, but also calculates its global LOGLOG address set presentation. Assuming $Rev_j = \{frag_{j\_0}, frag_{j\_1}, frag_{j\_2}, \cdots, frag_{j\_(rj-1)}\}$ as the fragments sequence received by monitor $j$ where $rj$ is the number of fragments monitor $j$ receives, $P(frag_{j\_i})$ $(0 \leq$

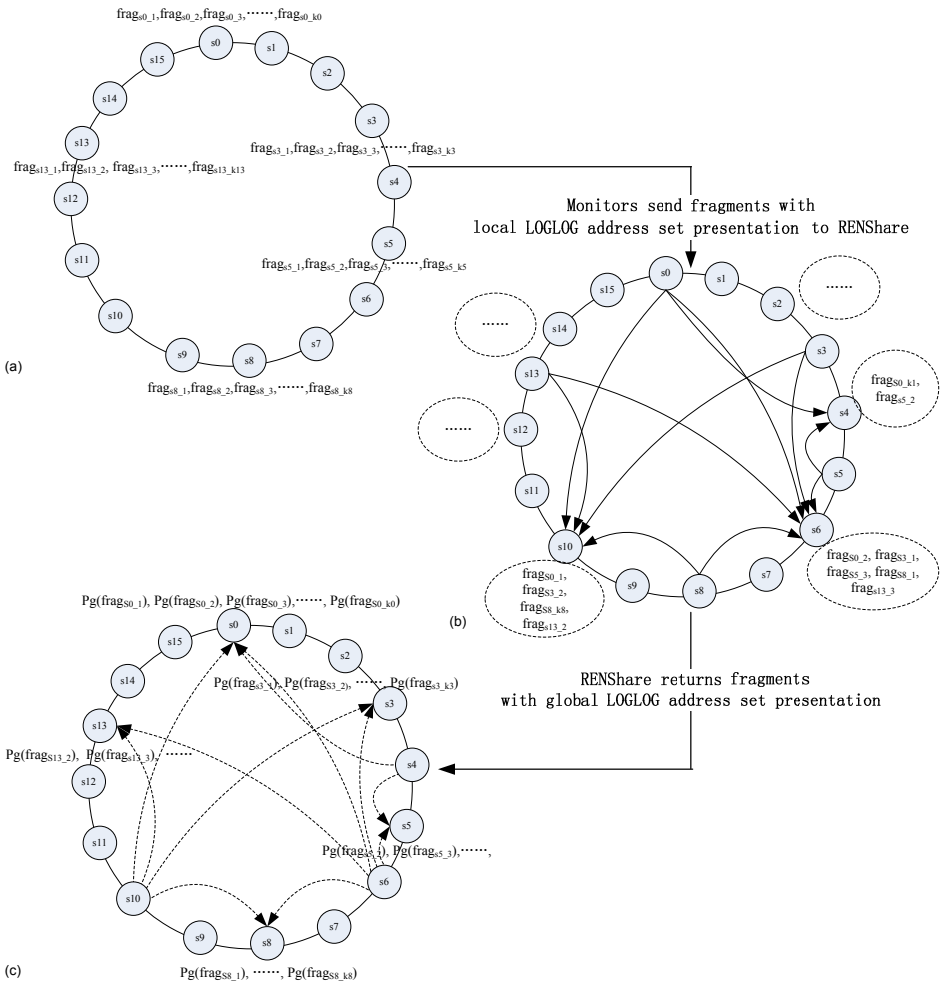



**Figure 3. The Functionality of RENShare**

$i \leq rj$) as the local LOGLOG address set presentation carried by fragment $frag_{j\_i}$, and $monitor(frag_{j\_i})$ $(0 \leq i \leq rj)$ as the IP address of source monitor which submit fragment $frag_{j\_i}$ to RENShare. Monitor $j$ uses Algorithm 2 to compute global LOGLOG address set presentation of fragments in $Rev_j$ to gain global LOGLOG address set presentation set $Pset_j$ and source monitor set $SrcIPset_j$ for distinct fragments in $Rev_j$. Lastly, monitor j runs Algorithm 3 to return the global LOGLOG address set presentation of each distinct fragment in $Pset_j$ to its corresponding source monitor set. As a result, all monitors know the global LOGLOG address set presentation of their representative fragments, as Figure 3(c) describes, $P_g(frag_{si\_h})$ presents the global LOGLOG address set presentation of fragment $frag_{si\_h}$.

Algorithm 2 describes how to calculate the global LOGLOG address set presentations for distinct fragments in $Rev_j$ at monitor $j$ $(0 \leq j \leq (n-1))$. In Algorithm 2, the function **max-merge**($\mathbf{loglog_v.P_g}$, $\mathbf{P(frag_{j\_v})}$) is described in Algorithm 4.

RENShare, like traditional DHT network, has good load-balanced manner [24]. Each monitor in RENShare undertakes limited computation and communication overhead; therefore, RENShare also has well scalability and can easily catch the exponentially growing speed of malware samples.

**Algorithm 2** RenShare accumulating algorithm in root monitor $j$

**Input:** subsignature set: $Acc_j = \{subsig_{j\_0}, subsig_{j\_1}, \cdots, subsig_{j\_rj}\}$.
**Output:** messages sent to source monitor set.

```
Initialize an empty source monitor set List MonitorSetList = NULL.
for (v = 0; v < rj; v++) do
    if can't find from MonitorSetList a monitor set node whose indicator equals to subsig_{j_v} then
        allocate a monitor set node MonitorSet_v: MonitorList = NULL, a fragment indicator indicator = subsig_{j_v} and a global attribute value attr = attr(subsig_{j_v}).
        add monitor(subsig_{j_v}) to MonitorSet_v.MonitorList.
        Add the monitor set node MonitorSet_v to MonitorSetList.
    else
        find from MonitorSetList a monitor set node MonitorSet_v whose indicator equals to subsig_{j_v}
        add monitor(subsig_{j_v}) to MonitorSet_v.MonitorList.
        update MonitorSet_v.attr = MonitorSet_v.attr + attr(subsig_{j_v}).
    end if
end for
for Each MonitorSet_u ∈ MonitorSetList do
    for Each monitor ID ID_w ∈ MonitorSet_u.MonitorList do
        send MonitorSet_u.MonitorList and MonitorSet_u.attr to monitor ID_w
    end for
end for
```

### 2.6 Heuristic Fragments Correlation

Heuristic fragments correlation module heuristically recognizes the common fragment of a family and then gains the global dispersion of the family. Ideally, the common fragment set of a family is behavior fragments shared by all the programs of the family, and is the signature of the family. Intuitively, source IP address set of a family is calculated by intersecting the source IP address set of fragments in the common fragment set. And the global dispersion of a family is the cardinality of its source IP address set. However, it is difficult to identify which programs belong to a family, let alone to gain the common fragment set shared by those programs. But, as malicious programs (samples) of a family are widespread, hence, the set of fragments that has large cardinality is more likely to be invariant behaviors if the program is malicious. Therefore, heuristic rules of recognizing the common fragment set are as follows: 1) maximize the cardinality of the fragment set, a.k.a. larger the cardinality of the fragment set is, the more likely the fragments in the fragment set are invariant behaviors of a family; 2) the common fragment set is specific, a.k.a. it is rarely seen in benign programs (low false positive). A program is said to match a set of fragments if its system calls sequence contains any fragment in the set. Assuming a set of fragments $fragSet$ and a program set $ProSet = \{Pro_1, Pro_2, \cdots\}$, the program set matched $ProSet$ from $ProSet$ is defined as $ProSet_{fragSet} = \{Pro_i \mid Pro_i \in ProSet$ and $Pro_i$ match $fragSet\}$.

**Definition 2: Heuristic Fragments Correlation Problem**
**INPUT**: the fragments set of a suspicious program $susPro_i$ returned by RENShare: $fragSet_{susPro_i}$; the global LOGLOG address set presentation for each fragment $j$: $P_g(j)(\forall j \in$

**Algorithm 3** Monitor $j$ return the global LOGLOG address set presentation to source monitors of received fragments

**Input:** Global LOGLOG address set presentation set $Pset_j$ and source monitor set $SrcIPset_j$

**Output:** Messages sent to source monitors of received fragments

```
for IP address node AD_v ∈ SrcIPset_j do
    find from Pset_j the LOGLOG node loglog_v whose indicator equals to AD_v.indicator
    for Address IP_w ∈ AD_v.AddressList do
        Return message containing AD_v.indicator and loglog_v.P_g to source monitor IP_w
    end for
end for
```

**Algorithm 4** The function **max-merge**

**Input:** Global LOGLOG address set presentation $P_g$, the local LOGLOG address set presentation $P_l$ to be merged, and $m$

**Output:** Global LOGLOG address set presentation $P_g$ after merging $P_l$

```
for (v = 0; v < m; v++) do
    if P_g.counter[v] < P_l.counter[v] then
        P_g.counter[v] = P_l.counter[v].
    end if
end for
```

$fragSet_{susPro_i}$) ; a benign programs pool: $PoolSet = \{Pro_1, Pro_2, \cdots\}$; and a threshold of false positive: $\theta_{FP}$.

**OUTPUT**: a common fragments set $CommSet$ for the suspicious program $SusPro_i$ such that satisfies as Formula 3.

$$\begin{aligned} &\textbf{Maximize} \quad \|P_{family}\| = \textbf{LOG\_intersect}_{j \in CommSet} P_g(j) \\ &\textbf{subject to} \quad \begin{cases} FP(CommSet) = \frac{|PoolSet_{CommSet}|}{|PoolSet|} \leq \theta_{FP} \\ CommSet \subseteq fragSet_{susPro_i} \end{cases} \end{aligned} \tag{3}$$

Seen from Formula 3, heuristic fragments correlation problem is transformed to a mathematical programming problem. In Formula 3, $\|P_x\|$ is the cardinality estimation of the LOGLOG address set presentation $P_x$, and $|set_x|$ is the number of elements in set $set_x$. Let $P_{set} = \{P_g(j) | j \in CommSet\}$. The operation $\textbf{LOG\_intersect}_{j \in CommSet} P_g(j)$ computes the approximate cardinality of intersection of all address sets represented by LOGLOG address set presentation set $P_{set}$. And $\textbf{LOG\_intersect}_{j \in CommSet} P_g(j)$ is simply referred to as $\textbf{LOG\_intersect}(P_{set})$.

**Operation LOG_intersect**($P_{set}$). Let $P_{set}$ as $\{P_1^g, P_2^g, \cdots, P_i^g, \cdots, P_q^g\}$, and $addSets = \{A_1^g, A_2^g, \cdots, A_i^g, \cdots, A_q^g\}$ be a set of address sets of end devices, where $A_i^g$ is the corresponding address set which is presented by LOGLOG address set presentation $P_i^g$. And let $(P_s^g \bigsqcup P_t^g)$ denotes **max-merge**$(P_s^g, P_t^g)$. Operation **LOG_intersect**($P_{set}$) aims to approximately estimate the cardinality of $(\bigcap_{i=1}^q A_i^g)$ from $P_{set}$. Formula 4 describes the inclusion and exclusion principle. Based on this principle and **max-merge** operation, we can directly calculate **LOG_intersect**($P_{set}$) as Formula 5.

$$\begin{aligned} |\bigcap_{i=1}^q A_i^g| \quad &= \sum_{i=1}^q |A_i^g| - \sum_{i,j:1 \leq i < j \leq q} |A_i^g \cup A_j^g| + \\ &\sum_{i,j,k:1 \leq i < j < k \leq q} |A_i^g \cup A_j^g \cup A_k^g| - \cdots + (-1)^{q-1} |\bigcup_{i=1}^q A_i^g| \end{aligned} \tag{4}$$

$$\begin{aligned}\mathbf{LOG\_intersect}(P_{set}) &= \sum_{i=1}^{q} \|logA_i^g\| - \\ &\sum_{i,j:1\le i<j\le q} \|logA_i^g \bigsqcup logA_j^g\| + \\ &\sum_{i,j,k:1\le i<j<k\le q} \|A_i^g \bigsqcup A_j^g \bigsqcup A_k^g\| \\ &- \cdots + (-1)^{q-1} \| \bigsqcup_{i=1}^{q} (logA_i^g)\|\end{aligned} \tag{5}$$

There is only one small problem with operation $\mathbf{LOG\_intersect}(P_{set})$. Since $\mathbf{LOG\_intersect}(P_{set})$ calculated based on multiple separate cardinality estimations of LOGLOG address set presentations and each estimation introducing an additive bias, $\mathbf{LOG\_intersect}(P_{set})$ will combine multiple biases. Reducing the bias of operation $\mathbf{LOG\_intersect}(P_{set})$ is our future work.

As a preliminary paper for SCMA malware detection system, we use IBM ILOG CPLEX Optimizer [19] to solve the heuristic fragments correlation problem. IBM ILOG CPLEX Optimizer is a high-performance mathematical programming solver. In future works, we will design a more efficient and effective algorithm for this problem.

### 2.7 Malicious Program Decision Module

Malicious program decision module decided whether the program is malicious or not based on the global dispersion of its family, which is deduced from the common fragments set of the program. Global dispersion of its family is as follows: $\|P_{family}\| = \|\mathbf{min\_intersect}_{j \in CommSet} P_g(j)\|$ where $CommSet$ is the common fragments set of the program. Here, we utilized a threshold based method: giving a empirical threshold of dispersion $\theta_P$, the program is malicious if the dispersion of its family greater than $\theta_P$.

## 3 Evaluation

In this section, we set fragment length $q$ as 10,a.k.a., fragment is system call subsequence with 10 system calls, and set the number of counters for LOGLOG address set presentation. We firstly prove the advantage of sharing fragments among monitors by comparing detection time between our SCMA and isolated centralized system. Then, comparing with existing distributed system, our SCMA can detect various malware including the ones using sophisticated technologies, let alone simple monomorphic malware.

### 3.1 Comparing with Isolated Centralized System

We select Conficker (Downadup) as the real malware for evaluation because Conficker spreads over the network with least human interaction: it first scans random networks to find new victims and if it infects a host successfully, it scans neighbor networks of the host to find victims nearby [14]. Setup the testbed for other malwares spreading using drive-by-download [15] or pay-per-install [16] methods is a very difficult task, because complex human interaction is needed. We setup a testbed with 10 LANs. each LAN contains 10 vulnerable end-hosts that will be infected by Conficker and other 5 normal end-hosts that will not be infected by Conficker. Each end-host is actually a virtual machine with Windows XP SP2 OS hosting in XEN hypervisor [20].

We compare collaborative SCMA monitors with isolated ones in terms of how timely they detect malicious programs, particularly the speed under various thresholds of dispersion .

For the sake of fair comparison, we deploy equal number of isolated and SCMA monitors (i.e. 5 monitors) in randomly selected LANs. The suspicious program discovering daemon is deployed on each vulnerable end-hosts of the selected LAN. The isolated monitors do not share fragments and their local LOGLOG address set presentations with other monitors and detect malicious programs only based on their own local LOGLOG address set presentations.

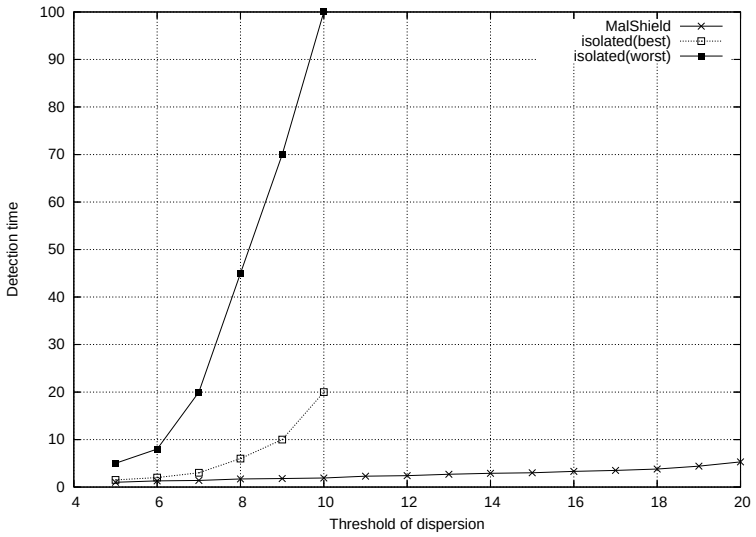



**Figure 4. Detection time vs thresholds of dispersion**

Figure 4 shows the detection time as a function of different thresholds of dispersion. From the figure, we can see that SCMA monitors can detect the malwares much faster than isolated monitors. If conducting the experiment in a large real network, the gap of detection speed between the two kinds will be much more larger. We can imagine that SCMA monitors can detect the malwares orders of magnitude faster than isolated monitors.

### 3.2 Comparing with Existing Distributed System

In this section, we select both simple mono-morphic and sophisticated malware as real malware. simple mono-morphic malware are early malware, including codered, Mydoom, and agobot. sophisticated malware are malwares using sophisticated technologies, including Conficker, and Storm.The testbed is as same as Section 3.1: 10 LANs of which each has 10 vulnerable end-hosts for corresponding malware and other secure 5 end-hosts.

WormShield [12] as the excellent presentation of existing distributed systems, we compare our SCMA with WormShield to verify the effectivity of SCMA. In WormShield we set the global threshold of dispersion both for source and destination IP address as 20, and set thresholds of dispersion in SCMA 20 too.

Table 1 describes detection ability of SCMA and WormShield. SCMA can detect all 5 malwares by using behavior fragments. while WormShield can only detect 3 mono-morphic worms, but can't detect sophisticated malwares for it using single substring of system call sequence, Seen form Table 1, SCMA is much more effective than existing distributed systems.

**Table 1. Detection Ability Comparison of our SCMA and WormShield**

| Malware | SCMA | WormShield |
|---|---|---|
| | detectable | detectable |
| codered | ok | ok |
| Mydoom | ok | ok |
| agobot | ok | ok |
| Storm | ok | no |
| Conficker | ok | no |

## 4 Conclusion and Future Works

In this paper we introduce a novel collaborative malware detection system, SCMA, which can detect various kinds of malware and is scalable, collaborative, and privacy-preserving. SCMA utilizes the three basic intrinsic characteristics of most malwares: 1) samples are automatic programs with network interaction; 2) samples of the same malware behave similarly; 3) samples of the same malware are widespread. SCMA puts a very high barrier behead of attackers. SCMA is host-network cooperated system that suspcious programs are discovered at host and malicious characteristics are validated at network level. SCMA is a first-of-its-kind real-world prototype system of such design and it shows a very promising direction for defeating malwares in this important battle.

SCMA is our preliminary work in host-network cooperated, distributed, and spatial malware detection. There are a lot of works to do. Firstly, we will do extensive experiments to validate our method: using more malwares especially new emerging ones as dataset; conducting the experiment in a large real network by cooperating with security companies, such as Qihoo or symantec. Secondly, we will explain in theory why the heuristic fragments correlation scheme is reasonable. Thirdly, although SCMA has certain degree of privacy-preserving because monitor in SCMA only know behavior fragments that routed to it according to DHT and only know the hash presentation of addresses of hosts, however, privacy-preserving is the foundation for various organizations join together to collaboratively detect malware, keeping privacy-preserving is another future work. Last but not least future work is making SCMA still effective facing some malicious monitors in the system; we will use algorithms in peer-to-peer (P2P) reputation systems [28] or trustworthy group making algorithm [27] to weight between monitors.

Another line of future work is to analyze the malware samples detected by SCMA. First, we cluster the samples of the same malware together, using some effective clustering schemes, such as spectral clustering [25] or multi-relational data mining [26]. Second, we generate the behavior signature composed of system calls subsequences that are common in samples for each cluster.

**Acknowledgements**

This work is supported by Program for Changjiang Scholars and Innovative Research Team in University (No.IRT 1012), the National High Technology Research and Devel-

opment Program of China (863 Program) (No. 2011AA01A103), the National Science Foundation of China (No. 61103194), and the National Science Foundation of China (No. 61003303). We appreciate anonymous reviewers for their valuable suggestions and comments.

## References

[1] http://en.wikipedia.org/wiki/Malware, *wikipedia.*, May, 2012.

[2] Clemens K., Paolo M. C.i, Christopher K., Engin K., Xiaoyong Z., Xiaofeng W., effective and efficient malware detection at the end host, *USENIX Security'09*, 2009.

[3] Internet Security Threat Report V17. http://www.symantec.com/business/threatreport/, *Symantec.*, June, 2012.

[4] http://maliciousnetworks.org/index.php. Mar. 2012.

[5] http://www.dshield.org/. Mar. 2012.

[6] Phillip P., Hassen S.I, AND Vinod Y., An Analysis of Conficker's Logic and Rendezvous Points, *SRI International Technical Report*, 2009.

[7] Guofei G., Phillip P., Vinod Y., Martin F., and Wenke L., BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation, *USENIX Security'07*, 2007.

[8] Guofei G., Junjie Z., and Wenke L., BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic, *15th Annual Network & Distributed System Security Symposium*, 2008.

[9] Guofei G., Roberto P., Junjie Z., and Wenke L., BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection, *USENIX Security'08*, 2008.

[10] Roberto P., Wenke L., and Nick F., Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces, *USENIX NSDI'10*, 2010.

[11] Jiyong J., David B., Shobha V., BitShred: Feature Hashing Malware for Scalable Triage and Semantic Analysis, *ACM CCS'11*, 2011.

[12] Min C., Kai H., Jianping P., and Christos P., WormShield: Fast Worm Signature Generation with Distributed Fingerprint Aggregation, *IEEE Transactions on Dependable and Secure Computing*, 4(2007), 88-104.

[13] H.-A. Kim and B. Karp, Autograph: Toward automated, distributed worm signature detection, *USENIX Security'04* , 2004.

[14] Eric Chien, Downadup: Attempts at Smart Network Scanning, *Symantec corp.*.

[15] Junjie Z., Christian S. et al, ARROW: Generating Signatures to Detect Drive-By Downloads, *WWW'11*, 2011.

[16] J. Caballero, C. Grier, C. Kreibich and V. Paxson, Measuring Pay-per-Install: The Commoditization of Malware Distribution, *USENIX Security'11* , 2011.

[17] M. Durand and P. Flajolet, Loglog counting of large cardinalities, *11th Annual European Symposium on Algorithms*, 2003.

[18] S. Schleimer, D. S. Wilkerson, and A. Aiken, Winnowing: local algorithms for document fingerprinting, *ACM SIGMOD'03*, 2003, 76-85.

[19] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/, *IBM corp.*, Jan. 2012.

[20] http://www.xen.org/, Mar. 2012.

[22] Clemens K., Paolo M. C.i, Christopher K., Engin K., Xiaoyong Z., Xiaofeng W., effective and efficient malware detection at the end host, *USENIX Security'09*, 2009.

[23] Somayeh M., Mehmet A., Christoph B., Runtime Verification in Distributed Computing, *Journal of Convergence*, 2(2011), 1-10.

[24] Qiying W., Tingting Q., Satoshi F., A Two-Level Caching Protocol for Hierarchical Peer-to-Peer File Sharing Systems, *Journal of Convergence*, 2(2011), 11-16.

[25]Ulrike v. L., A tutorial on spectral clustering, *Statistics and Computing*, 17(2007), 395-416.

[26] Carlos R V., Fernando T O., Paulo S., Angelo C C., Adriano M C., Rogeria C G de S., Pedro L P C., MR-Radix: a multi-relational data mining algorithm, *Human-centric Computing and Information Sciences*, 2(2012).

[27] Ailixier A., Tomoya E., Makoto T., Trustworthy Group Making Algorithm in Distributed Systems, *Human-centric Computing and Information Sciences*, 2(2011).

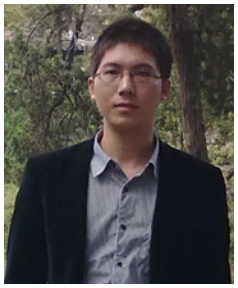
[28] Li X. and Ling L., PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities, *IEEE Transactions on Knowledge and Data Engineering*, 16(2004), 843-857.

[29] Manuel C., Jon C., Miguel C., Antony R., Lidong Z., Lintao Z., Paul B., Vigilante: End-to-end containment of Internet worm epidemics, *ACM Transactions on Computer Systems*, 26(2008), Article No. 9.

[30] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, A Survey of Mobile Malware in the Wild, *The 1st Workshop on Security and Privacy in Smartphones and Mobile Devices (CCS-SPSM'11)*, 2011.

[31] Georgios P., Philip H., Kostas A., Herbert B., Paranoid Android: versatile protection for smartphones, *ACM ACSAC'10*, 2010, 347-356.

[32] http://en.wikipedia.org/wiki/computer worm, *wikipedia.*, Jun. 2012.



**Huabiao Lu** rreceived the B.S. degree and M.S. degree from the National University of Defense Technology (NUDT) in 2006 and 2008, respectively, all in school of computer. He is a PhD candidate in Institute of Network and Information Security, National University of Defense Technology (NUDT) since March 2009. His current research interests are in malware detection, especially distributed and collaborative malware detection, and distributed computing.

**Xiaofeng Wang** is an assistant professor in School of Computer, National University of Defense Technology (NUDT), China. He completed his PhD at NUDT in 2009. His current research interests are in trust and security of networking systems, distributed and intelligent data processing. He has published several papers in renowned journals and conferences like IEEE/ACM CCGrid, AINA, IEEE Transactions on Services Computing and Elsevier FGCS etc.

**Jinshu Su** received the B. Sc degree in Mathematics from Nankai University in 1983, the M.S. degree and PhD degree in Computer Science, NUDT in 1989 and 1999, respectively. He is a full professor, the head of the Institute of network and information security, NUDT, and the academic leader of the State Innovative Research Team in University ("Network Technology") awarded by the Ministry of Education, CHINA. He has lead several national key projects of CHINA, including national 973 projects, 863 projects and NSFC Key projects. His research interests include high performance routers, high performance computing, wireless networks and information security. He is a member of the ACM and IEEE.