A Multigranularity Forensics and Analysis Method on Privacy Leakage in Cloud Environment

Deqing Zou, Jian Zhao, Weiming Li, Yueming Wu, Weizhong Qiang, Hai Jin, *Senior Member, IEEE*, Ye Wu, and Yifei Yang

***Abstract*—The problem of cloud forensics aims at processing multidimensional, massive, and heterogeneous data to collect and recover evidence in cloud environment. Existing approaches focus on excavating all suspicious behaviors from data and ignore privacy leakage details and behavioral characteristics. In order to conduct privacy leakage analysis in cloud specifically, we propose a multigranularity privacy leakage forensics method to analyze privacy violations caused by malware in cloud environment. By simulating the target virtual machine environment, our method can detect privacy leakage behaviors of malware without touching user's privacy data. We combine continuous RAM mirroring technology and dynamic taint analysis to assist the forensics investigation. To demonstrate the efficacy and utility of our method, we evaluate its performance with some real-world malware samples by comparing with some state-of-the-art malware analysis systems. Experimental results indicate that our method can identify more privacy leakage paths and behaviors.**

***Index Terms*—Cloud computing, cloud forensics, malware, privacy violation.**

I. Introduction

WITH the rapid development of the Internet of Things (IoT) technology, it has become one of the research hotspots in both industrial and academic area and the IoT devices have penetrated into people's life. Cloud computing, as the key technology of IoT, it does provide a powerful storage ability and computing power for IoT. However, with the particularity of the IoT equipment, it carries a large number of user's privacy data [1], which would cause a great loss if leaked out by malicious programs [2]. Therefore, a great concern should be paid in cloud-based IoT on the protection and leakage prevention of privacy data [3], [4]. At the same time, it also puts forward higher requirements to cloud forensics.

The privacy disclosure crimes in cloud-based IoT environment demands more accurate and precise forensics techniques to analyze malicious activities. One typical paradigm of cloud forensics follows four stages of operations [5]: 1) evidence source identification and preservation; 2) forensic data collection; 3) forensic data examination and analysis; and 4) evidence reporting and presentation. There exist many sophisticated methods for forensics investigation in cloud environment, such as the log-based approaches [6]–[10], the product-based approaches [5], [11]–[13], and the virtual machine introspection (VMI)-based approaches [14]–[17]. However, these techniques have not paid enough attention to information leakage caused by malware, especially ransomware and spyware.

In early 2017, the rise of global ransomware reached a new level of sophistication [18]. For example, by exploiting a vulnerability CVE-2017-0144 [19], WannaCrypt [20] had affected more than 100 000 organizations and institutions in nearly 100 countries around the world including schools, railway stations, self-service terminals, postal, gas stations, hospitals, government terminals, and so on. The destructive behaviors of vulnerabilities have induced serious leakage problem of privacy data. In particular, Wang *et al.* [21] focus on the information leakage caused by "Heart Bleed" over-read vulnerability [22]. They propose a quantitative risk measurement of information leak through buffer over-read. Keyloggers have been reported stolen credentials including SMTP, FTP, or remote databases [23]. Attacks on privacy information in cloud environment are becoming prevailing and damaging. Therefore, it is decisive to pay more attention to privacy leakage problems during the process of forensic analysis.

In the cloud computing environment, the issue of user privacy protection and privacy violation forensics has always been a challenge [24], because privacy leakage clues are easy to be destroyed and privacy violation behaviors are difficult to track and trace. The typical Internet infrastructure only has basic access control system and nonstandardized logging system, and the design of these mechanisms is just for resource use and business debugging, which is far from the purpose of computer forensics. For the issues mentioned above, in this

Manuscript received January 22, 2018; revised April 9, 2018; accepted May 3, 2018. Date of publication May 21, 2018; date of current version May 8, 2019. This work was supported in part by the National Key Research and Development Plan of China under Grant 2017YFB0802205, in part by the National Science Foundation of China under Grant 61370230, and in part by the Shenzhen Fundamental Research Program under Grant JCYJ20170413114215614. *(Corresponding author: Weiming Li.)*

D. Zou is with the Services Computing Technology and System Laboratory, Big Data Technology and System Laboratory, Cluster and Grid Computing Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and also with the Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen 518057, China.

J. Zhao, Y. Wu, W. Qiang, and H. Jin are with the Services Computing Technology and System Laboratory, Big Data Technology and System Laboratory, Cluster and Grid Computing Laboratory, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China.

W. Li is with the Network and Computing Center, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: lwm@hust.edu.cn).

Y. Wu and Y. Yang are with the Baidu Security Department, Baidu Online Network Technology (Beijing) Company Ltd., Beijing 100085, China.

Digital Object Identifier 10.1109/JIOT.2018.2838569

paper, we aim to design a novel forensics method to achieve three goals.

1) To analyze user's privacy leaks in cloud environment caused by malware without touching user's privacy.
2) To identify multiple privacy leakage paths and behaviors.
3) To do forensics on malware behaviors during its lifecycle.

Our system detects privacy leaks by adopting a multigranularity forensics technique. We propose a new method to focus on the propagation of user's privacy information during malware's lifecycle. For the purpose of protecting integrity and security of user's private information, we intercept the executable binaries from the cloud environment and simulate the target virtual machine environment with VMI technique. Based on the achievements of the virtual environment, our forensic method can investigate the criminal activities by integrating continuous RAM mirroring technology and dynamic taint analysis. By analyzing the experimental results, it is proved that our system can detect more privacy leakage paths and behaviors.

In summary, our contributions in this paper are as follows.

1) We present a privacy detection system that combines continuous RAM mirroring technology and dynamic taint analysis to facilitate forensic investigation in cloud environment, with which we can detect privacy leakage behaviors caused by malware related to keystroke leak, sensitive files leak, and sensitive memory leak, and extract the complete privacy disclosure paths.
2) Our forensic process is conducted in a completely independent simulation environment, thus we could do forensic analysis on privacy violations in cloud environment without affecting the online cloud service and accessing user data.
3) We demonstrate the effectiveness of our system in detecting privacy leaks caused by malware. We elaborate experimental evaluation including privacy detection of ransomware and spyware. Compared to some state-of-the-art techniques, our findings show that the proposed system can excavate and detect more privacy leakage behaviors.

The remainder of this paper is organized as follows. In Section II, we discuss the related work. In Section III, we introduce the overview and design details of our system. We present how to implement our system in Section IV, then test and evaluate our system in Section V and conclude this paper in Section VI.

II. Related Work

In this section, we discuss the previous works regarding cloud forensics. Majority cloud forensics approaches can be divided into three categories: 1) log-based approaches; 2) product-based approaches; and 3) VMI-based approaches.

A. Log-Based Approaches

In a cloud forensic investigation, an important step is to obtain the evidential data and take an analysis of the dataflow [25]. Logs are one of the most important evidences during the whole forensics process. Marty [6] provided a proactive approach on logging and accomplish a logging skeleton with some guidelines to ensure the completeness, accuracy, and readability of log information. In addition, in order to address issues of little evidence gathered by traditional static digital forensics techniques, Sibiya *et al.* [7] proposed and displayed a live digital forensics framework for a cloud environment by using data-mining techniques to fetch information from log files and employing artificial intelligence techniques to analyze network communication data. Zawoad *et al.* [8] first proposed secure-logging-as-a-service for collecting virtual machines' logs to construct a log database to assure the confidentiality of the cloud users, and then implement a secure logging architecture for network access logs in well-known OpenStack cloud platform. Additionally, Sang [9] designed a forensics-friendly system for the purpose of gathering log information from cloud computing faster and making some digital forensics easier and more convenient. By modifying Nova and Horizon to add a log acquisition module, Dykstra and Sherman [10] contributed three new forensics tools for the OpenStack which provide authentic access of virtual disks, API logs and guest firewall logs.

B. Product-Based Approaches

Some works attend to do forensics investigation of individual cloud products. Cho *et al.* [26] introduce Hadoop-based cloud forensic steps with remarkable guidelines including preparation, identification, live collection and analysis, statistic collection, transport, static analysis, and reporting. Meanwhile, a new research on forensics analysis of cloud storage services is proposed [11], which chooses four popular public cloud services, including Amazon S3, Google Docs, Dropbox, and Evernote to obtain artifacts of notable devices for investigating and analyzing. To validate the former work with respect to a new cloud forensics framework, Martini and Choo [12] conducted a detailed case study by using a popular open-source cloud storage product ownCloud [5]. They carry out a forensics analysis regarding both client and server of ownCloud and contribute the correlation of some artifacts to a forensics investigation. Similarly, by in-depth investigation on two popular cloud storage products including SkyDrive and Google Drive, Quick and Choo [13] grabbed some vital information of security concern that username and password can be determined from the conserved forensics images. Even in certain cases, the account password may be stored in plain text in hard drives or in memory captures or pagefile.sys files. With further investigation, a comparative analysis has been undertaken on forensics collection of Dropbox, Google Drive, and Microsoft SkyDrive, and it has been determined that the contents of files persist consistently during collection from the three well-known public cloud storage products.

C. VMI-Based Approaches

VMI is a technology that allows an observer to interact with a virtual machine client from outside. In 2003, Garfinkel and Rosenblum first presented a VMI-based intrusion detection system [27]. Since then, VMI techniques have

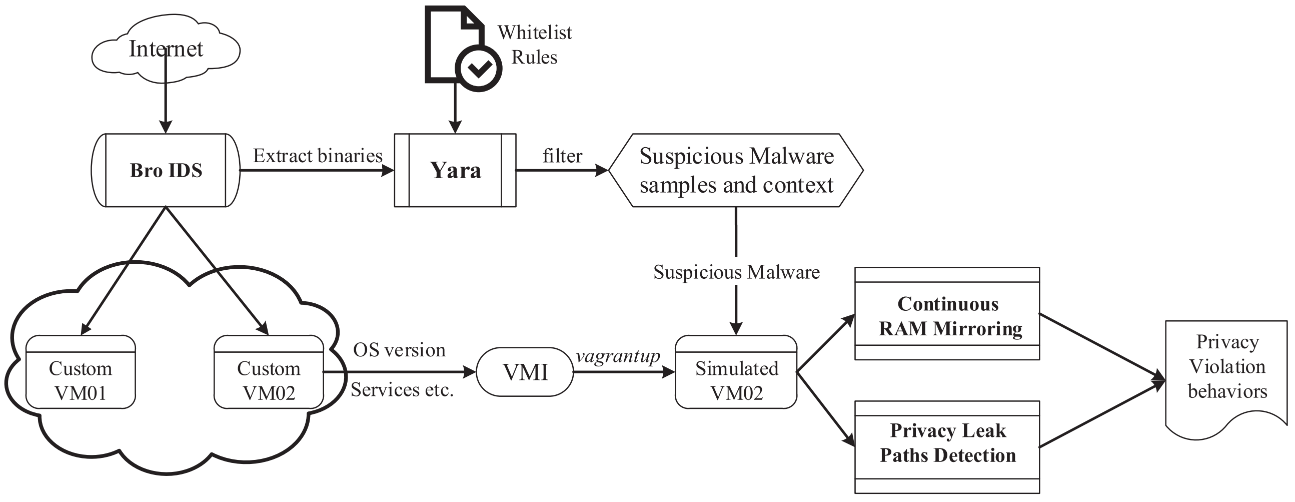


Fig. 1. Architecture of PVDS.

been widely known and used in various scenarios [16], [17]. Dykstra and Sherman [14] conducted an evaluation to measure the ability of some popular forensic tools such as EnCase Enterprise, AccessData FTK to extract data remotely from Amazon EC2. They also use the Eucalyptus cloud platform to inject an agent into the virtual machine to test these forensic tools by using VMI technique. Poisel *et al.* [15] detailed the potential of evidence collection from multiple virtualized environments in cloud forensics and describe the process on data acquisition from four different common hypervisors, namely XEN, KVM, VMware ESXi, and Microsoft Hyper-V.

In addition to the scheme mentioned above, some malware detection systems also have the functionality of privacy leakage detection. KingKong malware intelligent analysis system [28] is a set of fine-grained, highly transparent malware dynamic detection system provided by Institute of Software Chinese Academy of Sciences, it can find subtle abnormal behaviors by analyzing program's runtime information. Similarly, Habo [29] is a security platform developed by Tencent anti-virus lab, through which user can upload samples to obtain the basic information including suspicious behaviors, safety level and so on.

However, for log-based approaches, the contents of logs are usually very complicated, and it is not easy to extract privacy related contents from a vast amount of logs. Furthermore, the log-based forensics method is too coarse to accurately depict the privacy leakage behaviors of malicious softwares. Thus, our system adopted the multigranularity forensics analysis technology to achieve a more accurate privacy leak paths detection. The product-based forensics method can only be used for certain cloud environments with poor expansibility. However, our system aims at the general cloud environment, which is more widely applicable. For VMI-based forensics method, the concern is that excessive access to the user's virtual machine memory can lead to excessive performance loss. Our method only uses VMI technology to obtain config information of virtual machine when doing environment simulation, and then the forensics and analysis procedures will be conducted in the simulated environment without touching user's privacy data. By adopting continuous RAM mirroring and dynamic taint tracing analysis method, privacy violation detection system (PVDS) can do a multigranularity forensics investigation on privacy leakage in cloud environment.

III. System Desgin

In this section, we first describe the system architecture then introduce the four phases in details.

A. System Overview

We now introduce the proposed PVDS, a customized system for privacy detection caused by malware in cloud. PVDS can be divided into four phases, as depicted in Fig. 1. The first phase of PVDS is suspicious malware interception. We intercept executable binaries from cloud using intrusion detection system (IDS) Bro [30], and then adopt YARA [31] to identify and classify malware samples from the obtained binaries. IDS is a device or software application that monitors a network or systems for malicious activity or policy violations. Any malicious activity or violation is typically reported to an administrator. IDS Bro is a powerful software framework for network analysis, focusing on network security monitoring with high expansibility. Cloud environment simulation is our second phase, once a suspicious binary is discovered, the virtual machine which downloads the binary will be simulated to ensure the security and privacy of user information. Then the forensic investigation is commenced in the simulated environment. We adopt a coarse-grained forensic analysis method named continuous RAM mirroring technique to do forensics and analysis as our third phase. For the purpose of specialized analysis of privacy leakage paths, a fine-grained forensic method is introduced in the fourth phase to dynamically taint and track the propagation of user privacy. Finally, a report with respect to privacy leakage paths and behaviors is generated and retained as evidential data.

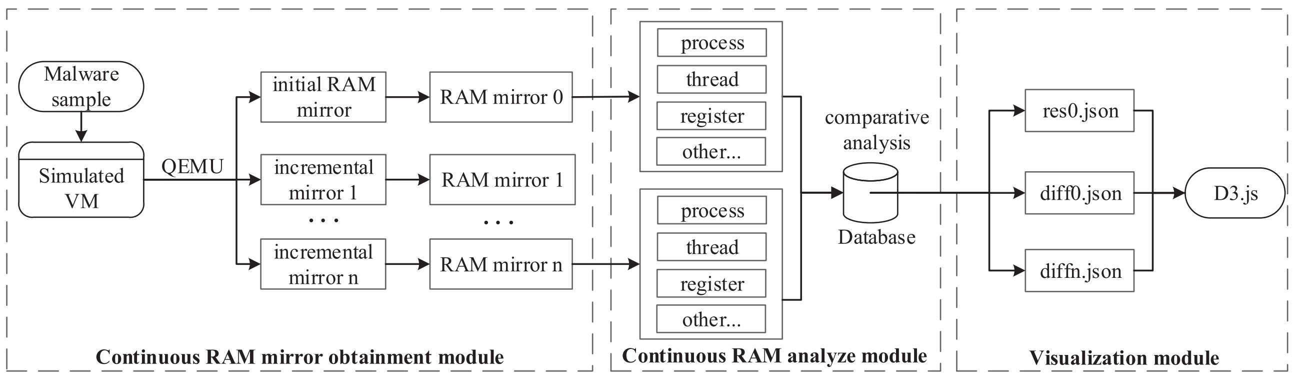


Fig. 2. Workflow of continuous RAM mirroring.

B. Suspicious Malware Interception

The first phase in PVDS is to intercept the suspicious malware from cloud environment which is the essential step for later forensic analysis. At first, we extract executable binaries from network packets by IDS Bro, then use YARA to filter suspicious application. We establish a whitelist of software which may appear in a cloud environment, the binaries that do not appear in the whitelist are identified as suspicious and should be further analysis.

C. Cloud Environment Simulation

To assure the integrity and security of user's privacy, we use Vagrant [32] to implement the simulation of the target virtual machine environment. Initially, we get the operating system version, services and applications of the target virtual machines with VMI technology. The operating system type is obtained by characteristic code and the version is determined by the offset of kernel debug structure, and finally traverse the kernel debug structure to find all running processes and modules in the system. Based on these information above, we could conduct a more pertinent analysis of suspicious malware. For example, if the running SSL version exists Heart Bleed vulnerability, a SSL private key leakage detection will be conducted.

A vagrant deployment script is automatically generated after the information is retrieved, and then the simulated virtual machine environment will be constructed. When detecting leakage of sensitive files, the simulated sensitive files would be copied into the simulated virtual machine.

D. Continuous RAM Mirroring

Continuous RAM mirroring is the proposed forensic method which based on the volatility framework [33]. By analyzing multiple RAM mirrors between different time intervals, the state changes of the system are acquired. Due to the storage overhead in the acquisition process of mirrors, continuous RAM mirroring adopts dirty page mechanism to continuously gain the modified part of the virtual machine RAM at different intervals, and then restores them based on the initial RAM mirror.

We divide continuous RAM mirroring into three modules, as shown in Fig. 2. First, we use the mounting method to copy the startup program and the malicious samples into the QEMU virtual machine. After starting the virtual machine, we save the virtual machine data including disk, memory and device status. And then we extract the physical memory block from these data to generate the initial RAM mirror. The acquire procedure of continuous RAM mirroring can be segmented into four phases: 1) at first, scan all memory pages of virtual machine and set as dirty pages; 2) next, save the previous dirty pages and mark modified memory page as new dirty pages; 3) then, store dirty pages as incremental RAM mirrors; and 4) finally, patch incremental RAM mirrors onto initial RAM mirror sequentially for analysis.

To dig specific runtime behaviors of suspicious malware, we conduct a detailed comparison analysis to extract the state changes between different RAM mirrors. In the comparative analysis, we analyze the state changes of analysis objects, such as process, registry, service, and network connection. The states of each analysis object are unchanged, changed, and new. According to the principle of these three states, we can determine the differences between two mirrors and derive the changes of the system.

In the end, the visualization module generates an easy-to-display JSON data format analysis report, and then D3.js (Data-Driven Documents) is adopted to implement our visualization module.

E. Privacy Leak Paths Detection

We define three privacy leak paths, i.e., keystroke leak, sensitive memory leak, and sensitive files leak. Keystroke leak means that the malware records and saves the user's keyboard input history. Sensitive memory leak means that some special memory regions are leaked by malware, such as those regions that contain host information, browsing records, private keys, etc. Sensitive files leak refers to malwares that access user's files and then send it to the Internet or write it to another encrypted file. For example, WannaCrypt encrypts user's files and ransoms bitcoins.

We adopt the whole system dynamic taint tracing method to do privacy leak paths detection because tainted memory related operations are often cross processes, through the operating system kernel to user mode. Moreover, malware often uses anti-debug techniques, which makes it difficult to be debugged

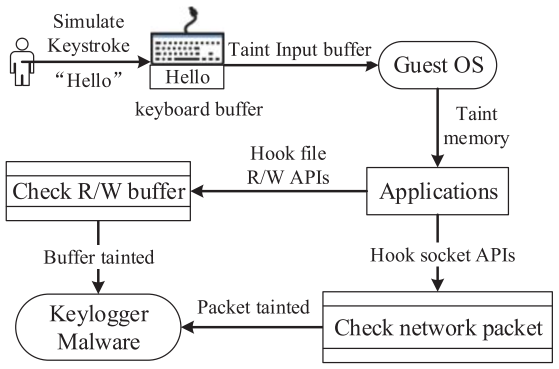


Fig. 3. Workflow of keylogger malware detection.

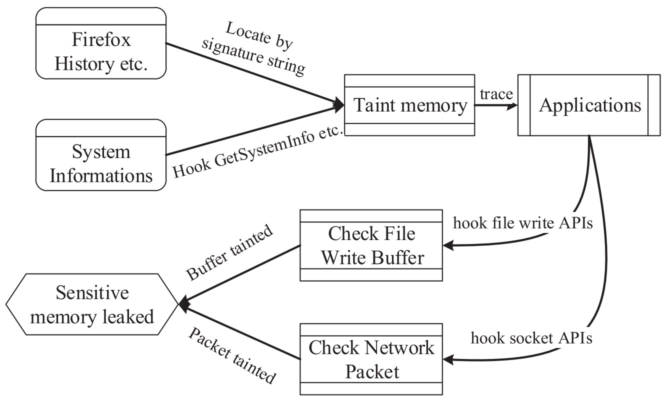


Fig. 4. Workflow of sensitive memory leakage detection.

and analyzed. Hence, the whole system dynamic taint tracing method can make our analysis more accurate.

In the following sections, we will describe our detection procedures of the three privacy leak paths in details.

1) Keystroke Leak: As shown in Fig. 3, by simulating keystroke events and tainting PS2 keyboard buffer, we use the tainted keyboard buffer as taint source to track the flow of the tainted data. When a process reads the tainted memory, the module name and the function name of the current program are determined by the function call shadow stack we maintained. In this case, we could conclude that the process has keylogger behaviors. In addition, we hook the file write function *WriteFile* in Kernel32.dll and Socket transmission functions in Ws2_32.dll, checking whether the buffer they use is tainted. If so, the target process has keystroke leakage behaviors.

We implement our privacy leak detection system based on the API hook framework provided by DECAF, we define sensitive information transmit functions set F_{trans} and privacy leak functions set F_{leak} as follows. As we set PS2 keyboard input buffer as our taint source, so keylogger detection has no F_{trans} set

$$F_{\text{leak}} = \left\{ \begin{matrix} \mathit{WriteFile}, \mathit{NtWriteFile}, \mathit{WSASend}, \\ \mathit{send}, \mathit{WSASendTo}, \mathit{sendto} \end{matrix} \right\}.$$

2) Sensitive Memory Leak: As shown in Fig. 4, we implement the automated sensitive memory tainting and system information tracing. In the first part, we implement Firefox history and cookie tainting, SSL private key tainting. We set these memory areas as taint source, tracking the flow of tainted

TABLE I
SIGNATURE STRINGS OF SENSITIVE MEMORY

Sensitive memory	**Signature string**	**Search domain**
Firefox history	'0x06 0x25 0x08'	physical memory
	'0x06 0x25 0x09'	
	'0x00 0x25 0x08'	
	'0x00 0x25 0x09'	
Firefox cookie	'0x04 0x06 0x06'	physical memory
	'0x05 0x06 0x06'	
SSL private key	'0x00 0x01 0x00 0x00'	address space of process ssh-agent
	'0x00 0x02 0x00 0x00'	
	'0x00 0x03 0x00 0x00'	
Chrome history	'0x08 0x00 0x01 0x06'	physical memory
	'0x08 0x00 0x02 0x06'	
	'0x08 0x00 0x03 0x06'	
	'0x09 0x00 0x01 0x06'	
	'0x09 0x00 0x02 0x06'	
	'0x09 0x00 0x03 0x06'	
USN records	'0xe2 0x80 0xaa'	physical memory
	'0xe2 0x80 0xab'	
	'0xe2 0x80 0xac'	
	'0xe2 0x80 0xad'	
	'0xe2 0x80 0xae'	
	'0xe2 0x80 0x8e'	
	'0xe2 0x80 0x8f'	
PUBLICKEY STRUCT	'0x07 0x02 0x00 0x00 0x00 0xa4 0x00 0x00'	physical memory
Lsa_key(WIN7)	'0x83 0x64 0x24 0x30 0x00 0x44 0x8b 0x4c 0x24 0x48 0x48 0x8b 0x0d'	address space of process lsass.exe
Lsa_key(WIN8)	'0x83 0x64 0x24 0x30 0x00 0x44 0x8b 0x4d 0xd8 0x48 0x8b 0x0d'	address space of process lsass.exe
Lsa_key(WIN10)	'0x83 0x64 0x24 0x30 0x00 0x48 0x8d 0x45 0xe0 0x44 0x8b 0x4d 0xd8 0x48 0x8d 0x15'	address space of process lsass.exe

memory to analyze whether sensitive memory flows to malware processes and whether they ware leaked or not. The key to implement automated sensitive memory tainting is sensitive memory locating which is implemented by pattern matching in VM memory.

Table I shows the signature string of sensitive memory, the signature string is the beginning character string of sensitive memory when they are loaded. We first intercept memory reading and writing instructions, and then search in VM memory to find the location of sensitive data when memory changes. As for SSL private key's location retrieving, we focus on RSA key. With the knowledge of RSA key layout, we search for the signature string and do validation. As the signature string is too simple which leads to many false positives, a further verification based on the RSA key layout is required. In rest of the Table I, the USN records contain file operation histories. The PUBLICKEYSTRUC is a windows cryptography struct, which signature string represents a public and private key pair. The Lsa_key means LsaInitializeProtectedMemory_KEY, it is the signature string which is used to retrieve windows password by malware mimikatz. The active user's encrypted password can be obtained directly from lsass.exe and the encryption algorithm is reversible. Thus, the plaintext password can be retrieved.

The system information tracing is implemented through DECAF API hook framework. We hook some system

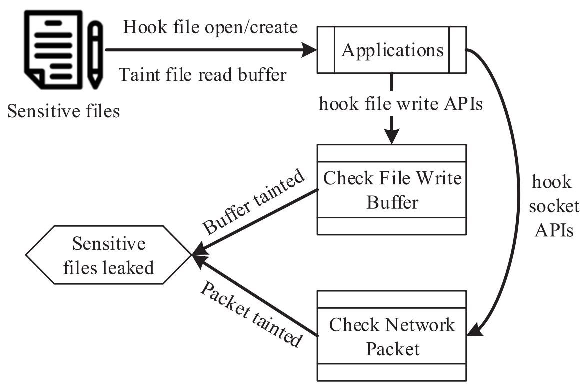


Fig. 5. Workflow of sensitive files leakage detection.

information related functions, such as *GetComputerName* and *GetSystemInfo*, and mark their parameters as taint sources to trace where system information goes.

For sensitive memory leakage detection, we define F_{trans} and F_{leak} as follows:

$$F_{\text{trans}} = \left\{ \begin{matrix} GetComputerName, \\ GetComputerNameEx, \\ GetSystemInfo, \\ GetUserName, \\ GetUserNameEx, \\ GetSystemDirectory, \\ GetWindowsDirectory, \\ ExpandEnvironmentStrings \end{matrix} \right\}$$

$$F_{\text{leak}} = \left\{ \begin{matrix} WriteFile, NtWriteFile, WSASend, \\ send, WSASendTo, sendto \end{matrix} \right\}.$$

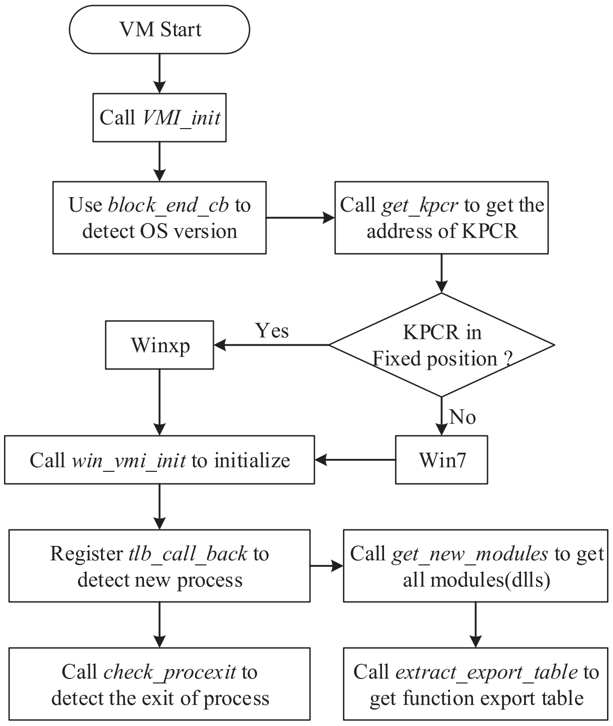
3) Sensitive Files Leak: As shown in Fig. 5, sensitive files leakage detection is implemented by hooking the file operation APIs. We hook *OpenFile* and *CreateFile* to get the name of opened files to identify whether it is a sensitive file, hook *ReadFile* function and mark its parameters as taint source to track the flow of tainted data. Finally, through the hook file write function and network transmission functions, we further confirm whether sensitive file contents are leaked out. For sensitive files leakage detection, we define F_{trans} and F_{leak} as follows:

$$F_{\text{trans}} = \left\{ \begin{matrix} OpenFile, CreateFile, ReadFile, \\ NtOpenFile, NtCreateFile, NtReadFile \end{matrix} \right\}$$

$$F_{\text{leak}} = \left\{ \begin{matrix} WriteFile, NtWriteFile, WSASend, \\ send, WSASendTo, sendto \end{matrix} \right\}.$$

IV. Implementation

In this section, we present implementation details of the key techniques in our system. In suspicious malware interception, we use IDS Bro and YARA to intercept suspicious executable programs from cloud environment. IDS Bro is a powerful network analysis framework focusing on network security monitoring. We extract executable binaries from http requests by bro filter scripts. As more and more websites are using https protocol to ensure session security, we need users to trust our root certificate to do https proxy, which makes suspicious malware interception module workable on https sessions.



VM Start
Call VMI_init
Use block_end_cb to detect OS version
Call get_kpcr to get the address of KPCR
KPCR in Fixed position ?
Yes
Winxp
No
Win7
Call win_vmi_init to initialize
Register tlb_call_back to detect new process
Call get_new_modules to get all modules(dlls)
Call check_procexit to detect the exit of process
Call extract_export_table to get function export table

Fig. 6. Workflow of our VMI framework.

We implement the continuous RAM mirroring module based on the Volatility framework. Volatility is an open source tool for memory forensics analysis. It can analyze the exported memory mirroring by deriving the kernel data structure and use the plugins to obtain the details of the memory and running status of the system. For instance, we employ its process plugin to detect process objects, and its connect plugin to find out network objects.

In privacy leak paths detection, we adopt the DECAF framework as our primary tool [34], [35]. DECAF is a dynamic binary analysis framework and provide precise trace tracking analysis and many event-driven interfaces which make our forensics and analysis more convenient. As shown in Fig. 6, we implemented a simple VMI framework based on DECAF. The OS version can be first determined based on the location of the KPCR, Then, the callback function of the translation block can be registered to detect the creation and exit of the processes. When a process is created, we traverse all the modules it loaded to get the function export tables of each module and then implement API hook framework based on these function export tables. We use the API hook framework to hook functions in F_{trans} and taint their parameters as taint source, and hook functions in F_{leak} to detect whether the tainted memory has been leaked.

V. Evaluation

In this section, we detail our evaluation results to validate the effectiveness of PVDS. First, we introduce our experimental datasets. Then we describe some state-of-the-art malware

TABLE II
BASIC INFORMATION OF OUR DATASET

Malware Type	Malware Sample	Detection results
Ransomware	CryptoWall	**Kaspersky**: Trojan-Ransom.Win32.Cryptodef.cng, **Microsoft**: Ransom:Win32/Crowti.A
	Satan	**Malwarebytes**: Ransom.Satan, **Ikarus**: Trojan-Ransom.Satan
	Jaff	**Kaspersky**: Trojan-Ransom.Win32.Scatter.ue, **Malwarebytes**: Ransom.Jaff
	Win32.Locky	**Kaspersky**: Trojan-Ransom.Win32.Locky.xvv, **Microsoft**: Ransom:Win32/Locky
	WannaCrypt	**Kaspersky**: Trojan-Ransom.Win32.Wanna.m, **Malwarebytes**: Ransom.WannaCrypt
	Cerber	**Microsoft**: Ransom:Win32/Cerber, **Malwarebytes**: Ransom.Cerber
	Ergop	**Kaspersky**: Trojan-Ransom.Win32.Gen.fcd, **Microsoft**: Ransom:Win32/Ergop.A
	Genasom	**Kaspersky**: Trojan-Ransom.Win32.Purgen.fu, **Microsoft**: Ransom:Win32/Genasom
	Jaffrans	**Kaspersky**: Trojan-Ransom.Win32.Scatter.vx, **Microsoft**: Ransom:Win32/Jaffrans.A
	Locky	**Kaspersky**: Trojan-Ransom.Win32.Locky.yji, **Microsoft**: Ransom:Win32/Locky
	Spora	**Microsoft**: Ransom:Win32/Spora, **Kaspersky**: Trojan-Ransom.Win32.Spora
	Teerac	**Microsoft**: Ransom:Win32/Teerac, **Malwarebytes**: Ransom.Crypt0L0cker
	Petya	**Kaspersky**: Trojan-Ransom.Win32.Petr.a, **Microsoft**: Ransom:Win32/Petya.A
Spyware	SpyNet	**Symantec**: W32.Spyrat, **Malwarebytes**: Backdoor.SpyNet
	Shifu	**Malwarebytes**: Spyware.Shifu, **Microsoft**: TrojanSpy:Win32/Skeeyah.A!rfn
	Keylogger	**Kaspersky**: Trojan-Spy.MSIL.KeyLogger.brse, **Antiy-AVL**: Trojan[Spy]/MSIL.KeyLogger
	Zbot	**Kaspersky**: Trojan-Spy.Win32.Zbot.ntpf, **Malwarebytes**: Spyware.Citadel
	Omaneat	**Microsoft**: TrojanSpy:MSIL/Omaneat.C, **Kaspersky**: Trojan.MSIL.Inject.epwf

detection tools to conduct a comparative analysis with PVDS. Finally, our coarse-grained forensic analysis and fine-grained privacy leakage detection results are reported. Our system is built on an Ubuntu OS 14.04 host with Intel Xeon E5-2630 v3 and 64 GB RAM, and the malware experiments are conducted on Windows VMs.

A. Datasets

Our malware datasets are mainly from malwaredb [36]. These malware samples are chosen from the current typical ransomware and spyware samples. As shown in Table II, we provide types of malware samples and detection results by some antivirus software vendors. Malware can be divided into corresponding families according to its behaviors, so we select a representative sample to do experiment for each malware family.

B. Evaluation Indicators

We choose two state-of-the-art malware detection tools, i.e., KingKong [28] and Habo [29], to do our comparative analysis for continuous RAM mirroring module.

In order to obtain more comprehensive privacy violation evidence, we define the detection attributes for continuous RAM mirroring according to the key events that may be involved in malware behaviors. Table III shows the detection attributes and our detection purpose. *sids* is used to get user's sid, which is security identifier for windows. *devicetree* can get the process tree of the system. *injections* can detect the vestiges of process injection. *privileges* plugin is used to detect privilege promotion events caused by malware, and *dlls* plugin is used to get the modules loaded by malware. we use *network* plugin to obtain the network connections created by malware and the *mutants* plugin is used to get the mutants created by malware which is often used as a feature to identify malware. we use *mftentries* plugin to detect file operation records of malware through master file table, which is the core of the NTFS file system. The *registry* plugin is used to get registry operation records of malware and *processes* plugin is used to detect processes created by malware.

TABLE III
DETECTION ATTRIBUTES AND PURPOSE

Detection attributes	Detection purpose
sids	Getting the user's sid
devicetree	Enumerating process trees
injections	Detection of process injection
privileges	Detect privilege promotions
dlls	Module loaded by malware
network	Malware network connections
mutants	Mutants created by malware
mftentries	Getting file operation records of malware
registry	Registry operation records
processes	Detect processes created by malware

Our system focuses on the detection of privacy violations caused by ransomware and spyware. File encryption and keystroke leakage are typical behaviors of these two malware types, so we use WannaCrypt and Keylogger samples to do detect evaluation in our experiment.

C. Results and Discussion

We first present our coarse-grained forensic analysis results. As shown in Fig. 7, we conduct a comparative analysis of detection results on these ten detection attributes reported by Habo, KingKong, and continuous RAM mirroring. During the experiment, we set the malware running time to 100 s, the incremental image acquisition interval to 500 ms, it takes about 20 h to analyze 200 GB RAM mirrors for each sample. The comparative analysis results show that continuous RAM mirroring outperforms KingKong and Habo, especially for attributes including sids, devicetree, injections, and privileges. KingKong and Habo analysis system fails to analyze these attributes, which are covered in our system with a more comprehensive and detailed analysis results. This allows us to provide more support to forensics and a wider coverage of privacy violation behaviors, which can validate the effectiveness of continuous RAM mirroring method and also provide a more accurate analysis report for our fine-grained analysis.

We use WannaCrypt sample to test our sensitive file leakage detection module and set up a private file test.txt in simulated

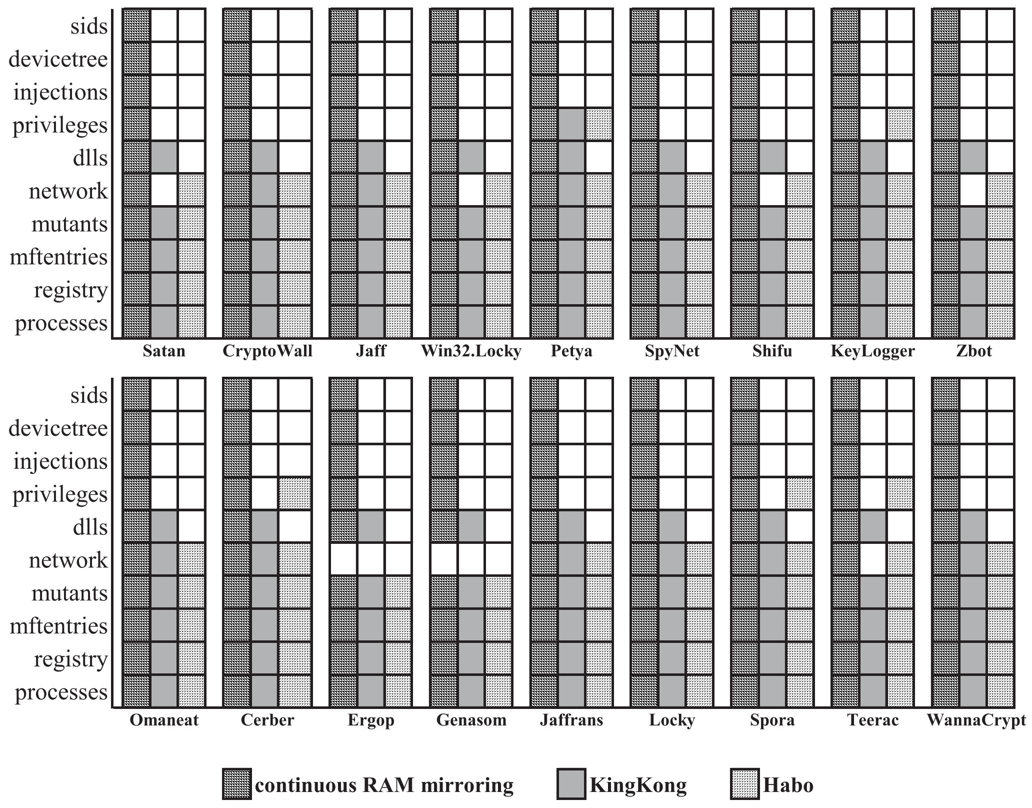


Fig. 7. Comparative analysis of continuous RAM mirroring, KingKong, and Habo.

TABLE IV
DETECTION LOG OF WANNACRYPT

WannaCrypt
Monitor CreateFile: Filename = test.txt
Target file open operation monitored !
. . .
Monitor ReadFile: Filename=test.txt, Read Buffer tainted
Monitor ReadFile: Filename=test.txt, Read Buffer tainted
. . .
Monitor CreateFile: Filename=test.txt.WNCRY
. . .
Monitor WriteFile: Filename=test.txt.WNCRY, Write Buffer tainted 4 Bytes
Target file contents leakage detected !

VM on its desktop. As shown in Table IV, the access operations to private file of this malware sample are captured by monitoring file open and create functions, and then monitor *ReadFile* function and set read buffer as taint source. Finally, we detect that the write buffer contains tainted data which is written to test.txt.WNCRY. These experimental results show that our technique can capture the privacy violation behaviors and encryption procure of this malware sample.

We use a keylogger malware sample *Spyware.KeyLogger* reported by Malwarebytes to illustrate the effectiveness of our system. We taint the keystroke dataflow with *notepad.exe* running in the foreground. As shown in Fig. 8(a), it shows the modules and functions involved in keystroke data flow. According to the normal keystroke dataflow, system process first gets the keystroke data from QEMU PS2 keyboard buffer, then flows to *csrss.exe*, and eventually flows to *notepad.exe*. Through the continuous RAM mirroring analysis results, it has been detected that the processes *WindowsFirewall* is generated by the keylogger malware sample. As shown in Fig. 8(b), it can be found that keystroke memory flows from PS2 keyboard buffer to malicious processes directly and the invoked functions have changed a lot. In view of the dataflow as shown in Fig. 8(b), it can be concluded that this sample has keylogger behaviors which change the dataflow of keystroke. Experimental results show that our system records the keystroke dataflow completely and can detect the keylogger malware effectively.

Additionally, we find that the malware sample Cerber has sensitive memory leak behaviors, which read memory data of *system.exe*, *csrss.exe*, *winlogon.exe*, and *lsass.exe*.

For privacy leakage paths detection module, the performance loss experiment of PVDS under different taint source sizes has been conducted in Windows XP VM with 2G RAM. As shown in Fig. 9, we calculate the average running time of MD5 algorithm and overload in different taint source size. The DECAF (no tainting) represents the program is running under DECAF environment with tainting disabled, and the PVDS represents the program is running under PVDS environment with sensitive files leakage detection plugin loaded. The MD5 algorithm has frequent memory read–write operations and a good taint memory diffusion effect, which can effectively illustrate the performance loss of our system. The experimental results show that the average performance loss of PVDS is

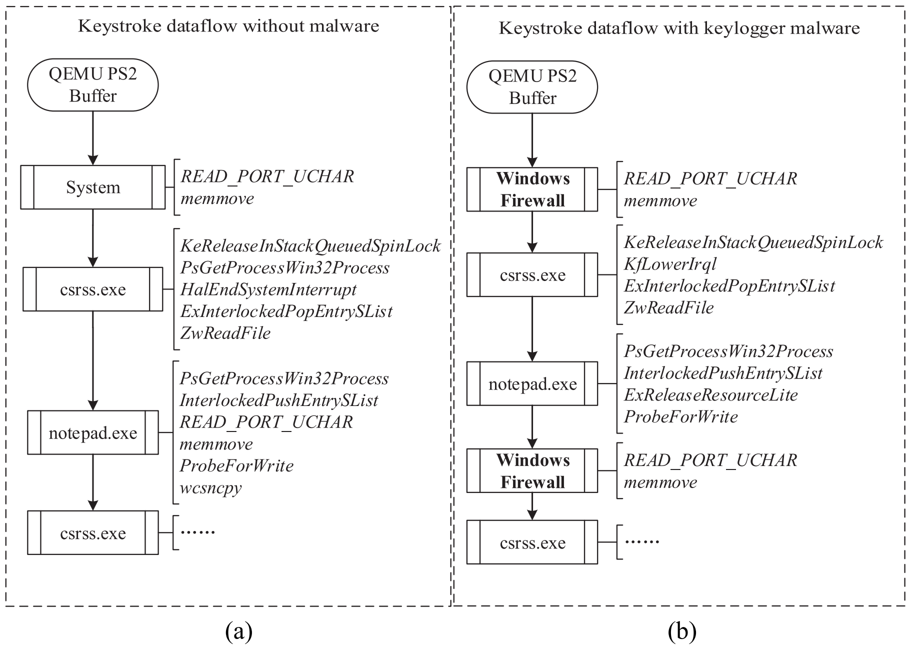


Fig. 8. Dataflow of keystroke with and without keylogger.

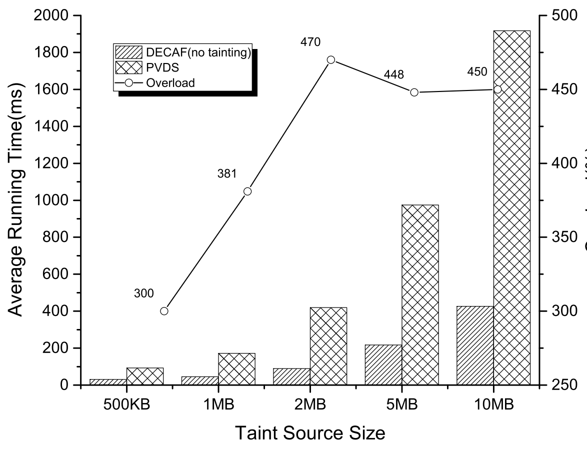


Fig. 9. Overload of PVDS in different taint source size.

about four times in different taint source size, which is less than six times mentioned in DECAF [34].

The above experimental results show that PVDS not only can detect privacy leak behaviors of malware on sensitive memory, sensitive files and keystroke, but also can retrieve the complete privacy leak paths. Moreover, PVDS provides a lower performance loss than DECAF. Thus, it can be concluded that PVDS can do forensics and analysis on privacy violations caused by ransomware and spyware effectively in cloud environment.

VI. Conclusion

This paper presented PVDS, a privacy violation detect system in cloud environment. Our system focuses on the propagation of user's privacy data and adopts a combination of continuous RAM mirroring technique and dynamic taint analysis to assist forensics investigation. We could do forensic analysis on privacy violations in cloud environment without affecting the online cloud service and accessing user's VM and private data. Our extensive experimental evaluation shows that PVDS could detect more privacy leakage paths and behaviors, especially regarding keylogger, sensitive file and memory.

Although evaluation results show the usability and effectiveness of our system in privacy leak detection, our system still suffers some limitations. For one hand, ten detection attributes in malware behaviors detection for continuous RAM mirroring are not enough. We would dig out more attributes to get clearer and more accurate privacy disclosure behaviors in the future. On the other hand, PVDS could do forensic analysis in keystroke leak, sensitive memory leak and sensitive files leak, but sensitive memory area we defined is limited and privacy leakage paths we defined cannot cover all privacy leakage situations. Thus, more privacy leakage paths need to be added in the future.

References

[1] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in Internet-of-Things," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1250–1258, Oct. 2017.
[2] B. Dorsemaine, J.-P. Gaulier, J.-P. Wary, N. Kheir, and P. Urien, "A new threat assessment method for integrating an IoT infrastructure in an information system," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, Atlanta, GA, USA, 2017, pp. 105–112.
[3] H. Jin, W. Dai, and D. Zou, "Theory and methodology of research on cloud security," *Sci. China Inf. Sci.*, vol. 59, no. 5, 2016, Art. no. 050105.
[4] M. Hossain, R. Hasan, and A. Skjellum, "Securing the Internet of Things: A meta-study of challenges, approaches, and open problems," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, Atlanta, GA, USA, 2017, pp. 220–225.
[5] B. Martini and K.-K. R. Choo, "Cloud storage forensics: OwnCloud as a case study," *Digit. Invest.*, vol. 10, no. 4, pp. 287–299, 2013.
[6] R. Marty, "Cloud application logging for forensics," in *Proc. ACM Symp. Appl. Comput.*, 2011, pp. 178–184.
[7] G. Sibiya, H. S. Venter, and T. Fogwill, "Digital forensic framework for a cloud environment," in *Proc. Int. Inf. Manag. Corporat.*, 2012, pp. 1–8.

[8] S. Zawoad, A. K. Dutta, and R. Hasan, "Seclaas: Secure logging-as-a-service for cloud forensics," in *Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Security*, 2013, pp. 219–230.

[9] T. Sang, "A log based approach to make digital forensics easier on cloud computing," in *Proc. 3rd Int. Conf. Intell. Syst. Design Eng. Appl.*, 2013, pp. 91–94.

[10] J. Dykstra and A. T. Sherman, "Design and implementation of FROST: Digital forensic tools for the OpenStack cloud computing platform," *Digit. Invest.*, vol. 10, no. 8, pp. S87–S95, 2013.

[11] H. Chung, J. Park, S. Lee, and C. Kang, "Digital forensic investigation of cloud storage services," *Digit. Invest.*, vol. 9, no. 2, pp. 81–95, 2012.

[12] B. Martini and K.-K. R. Choo, "An integrated conceptual digital forensic framework for cloud computing," *Digit. Invest.*, vol. 9, no. 2, pp. 71–80, 2012.

[13] D. Quick and K.-K. R. Choo, "Forensic collection of cloud storage data: Does the act of collection result in changes to the data or its metadata?" *Digit. Invest.*, vol. 10, no. 3, pp. 266–277, 2013.

[14] J. Dykstra and A. T. Sherman, "Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques," *Digit. Invest.*, vol. 9, no. 8, pp. S90–S98, 2012.

[15] R. Poisel, E. Malzer, and S. Tjoa, "Evidence and cloud computing: The virtual machine introspection approach,"*J. Wireless Mobile Netw.*, vol. 4, no. 1, pp. 135–152, 2013.

[16] L. Jia, M. Zhu, and B. Tu, "T-VMI: Trusted virtual machine introspection in cloud environments," in *Proc. 17th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2017, pp. 478–487.

[17] M. A. Kumara and C. D. Jaidhar, "VMI based automated real-time malware detector for virtualized cloud environment," in *Proc. Int. Conf. Security Privacy Appl. Cryptograp. Eng.*, 2016, pp. 281–300.

[18] *Ransomware Review of Microsoft*. Accessed: May 3, 2018. [Online]. Available: https://blogs.technet.microsoft.com/mmpc/2017/09/06/ransom-ware-1h-2017-review-global-outbreaks-reinforce-the-value-of-security-hygiene/

[19] *A Detailed Description of CVE-2017-0144*. Accessed: May 3, 2018. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144

[20] *WannaCry Ransomware Attack*. Accessed: May 3, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Wan-naCry_ransomware_attack

[21] J. Wang, M. Zhao, Q. Zeng, D. Wu, and P. Liu, "Risk assessment of buffer 'Heartbleed' over-read vulnerabilities," in *Proc. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, 2015, pp. 555–562.

[22] *The Heartbleed Bug*. Accessed: May 3, 2018. [Online]. Available: http://heartbleed.com/

[23] K. Thomas *et al.*, "Data breaches, phishing, or malware? Understanding the risks of stolen credentials," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1421–1434.

[24] J. Zhou, Z. Cao, X. Dong, and A. V. Vasilakos, "Security and privacy for cloud-based IoT: Challenges," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 26–33, Jan. 2017.

[25] K.-K. R. Choo, C. Esposito, and A. Castiglione, "Evidence and forensics in the cloud: Challenges and future research directions," *IEEE Cloud Comput.*, vol. 4, no. 3, pp. 14–19, Jan. 2017.

[26] C. Cho, S. Chin, and K. S. Chung, "Cyber forensic for hadoop based cloud system," *Int. J. Security Appl.*, vol. 6, no. 3, pp. 83–90, 2012.

[27] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. Netw. Distrib. Syst. Security Symp.*, 2003, pp. 191–206.

[28] *KingKong Malware Intelligence Analysis System*. Accessed: May 3, 2018. [Online]. Available: https://www.tcasoft.com:8443

[29] *Tencent Habo Analysis System*. Accessed: May 3, 2018. [Online]. Available:https://habo.qq.com/

[30] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, nos. 23–24, pp. 2435–2463, 1999.

[31] *YARA*. Accessed: May 3, 2018. [Online]. Available:https://virustotal.github.io/yara/

[32] *Vagrant*. Accessed: May 3, 2018. [Online]. Available: https://www.vagrantup.com/

[33] *Volatility*. Accessed: May 3, 2018. [Online]. Available: http://www.volatilityfoundation.org/

[34] A. Henderson *et al.*, "Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform," in *Proc. Int. Symp. Softw. Testing Anal.*, 2014, pp. 248–258.

[35] A. Henderson *et al.*, "DECAF: A platform-neutral whole-system dynamic binary analysis platform," *IEEE Trans. Softw. Eng.*, vol. 43, no. 2, pp. 164–184, Feb. 2017.

[36] *MalwareDB*. Accessed: May 3, 2018. [Online]. Available: http://malwaredb.malekal.com/index.php

Deqing Zou received the Ph.D. degree from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2004.

He is a Professor of computer science with HUST. His current research interests include system security, trusted computing, virtualization, and cloud security. He has applied for approximately 20 patents, authored or co-authored two books entitled *Xen Virtualization Technologies* (Huazhong Univ. Sci. Technol. Press, 2009) and *Trusted Computing Technologies and Principles* (Sci. Press, 2011) and over 50 papers, including papers published in the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON SERVICES COMPUTING, and the IEEE TRANSACTIONS ON CLOUD COMPUTING.

Dr. Zou has served as the PC Chair/PC member of over 40 international conferences.

Jian Zhao received the B.S. degree in information security from the China University of Geosciences (Wuhan), Wuhan, China, in 2015. He is currently pursuing the master's degree in cyber security at the Huazhong University of Science and Technology, Wuhan.

His current research interests include cloud computing security and vulnerability detection.

Weiming Li received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China in 2006.

He is currently an Associate Professor with the Network and Computing Center, Huazhong University of Science and Technology. He has authored or co-authored over 30 refereed papers. His current research interests include on system security, especially malware analysis and detection using binary analysis techniques, and network security.

Dr. Li was a recipient of the First Prize of the Hubei Province Science and Technology Progress in 2011.

Yueming Wu received the B.S. degree in information security from Southwest Jiaotong University, Chengdu, China, in 2016. He is currently pursuing the Ph.D. degree in cyberspace security at the Huazhong University of Science and Technology, Wuhan, China.

His current research interests include vulnerability detection and anomaly detection.

Weizhong Qiang received the Ph.D. degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2005.

He is an Associate Professor with the Huazhong University of Science and Technology. He has authored or co-authored approximately 30 scientific papers. His current research interest includes system security about virtualization and cloud computing.

Hai Jin (M'99–SM'06) received the Ph.D. degree in computer engineering from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1994.

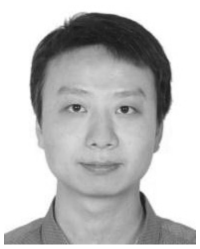
He was with the University of Hong Kong, Hong Kong, from 1998 to 2000, and as a Visiting Scholar with the University of Southern California, Los Angeles, CA, USA, from 1999 to 2000. He is a Cheung Kung Scholars Chair Professor of computer science and engineering with HUST. He is the Chief Scientist of the National 973 Basic Research Program Project of Virtualization Technology of Computing System. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz, Chemnitz, Germany. He has co-authored 22 books and over 700 research papers. His current research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.

Dr. Jin was a recipient of the Excellent Youth Award from the National Science Foundation of China in 2001. He is a member of the ACM.

Ye Wu received the Ph.D. degree in computer science from the Stevens Institute of Technology, Hoboken, NJ, USA.

He is a Principal Architect with Baidu Inc., Beijing, China, interested in access control security, security intelligence, and advanced applied cryptography for real applications. He with his team created Giano system that governs overall data and system security for Baidu IDC with the largest scale in Chinese Internet companies. He has authored or co-authored over 20 research papers.

Dr. Wu has presented keynotes and invited talks in major international academic conferences.



Yifei Yang received the master's degree in systems engineering from Xi'an Jiaotong University, Xi'an, China.

He is a Senior Architect with Baidu, Inc., Beijing, China. He is responsible for engineering research and development of Baidu's internal security products, including the largest sale security platform Giano. He was with Tencent, Inc., Shenzhen, China, building the security system for the cloud computing platform of Tencent.