A Method for Acquiring Network Information from Linux Memory Image in Software-Defined Networking

Shumian Yang[1,2,3], Lianhai Wang[1,2,3], Shuhui Zhang[1,2,3], Dawei Zhao[1,2,3], Lijuan Xu[1,2,3]

[1] Qilu University of Technology (Shandong Academy of Sciences), China
[2] Shandong Computer Science Center (National Supercomputer Center in Jinan), China
[3] Shandong Provincial Key Laboratory of Computer Networks, China
yangshm@sdas.org, wanglh@sdas.org, zhangshh@sdas.org, zhaodw@sdas.org, xulj@sdas.org

Abstract

Software defined network (SDN) is a novel network architecture which separates the control plane from the data plane of a network. Owing to its openness, programmability and centralized control, SDN accelerates the development of network technology. However, it also brings new security problems, such as SDN control security, external distributed denial of service (DDoS) attacks and the northbound-southbound interface security. Aiming at the various security attack problems in SDN, the physical memory forensic analysis method is applied to this new framework of SDN, which can extract and analyze the digital evidence including running status of the computer, the behaviour characteristics of the user, network information, opened file and register. The method in this paper mainly obtains the network information from the physical memory image file in real-time, including the address resolution protocol (ARP), network configuration information, and the network connection information. It does not depend on the kernel symbol table and system version. We have extracted the network information under a wide range of operating system versions. Finally, the method is verified on the ubuntukylin 14.04 system, by obtaining various network information, and the experiment results show that the method has high accuracy and effectiveness on comparing with the Volatility tool.

Keywords: Software-defined networking (SDN), Linux memory analysis, Software defined architecture security, Memory forensics

1 Introduction

Software-defined networking (SDN) is a revolutionary new network framework which achieves the separation of network control plane and the data plane [1]. While providing the centralized control and software programming, the network itself faces many security problems, and a series of corresponding protection strategies are proposed for the possible security risks in SDN. Security threats detection is the premise and key link of all the strategies. By analyzing the physical memory image file in real time, monitoring the running status of the network in real time, it actively discovers intrusion intentions and initiates corresponding security response to effectively improve the security performance of the entire network system. It is an important part of the network security problem.

Although, software-defined networking provides the separation of control level and data level, simplifies the process of the underlying hardware and network configuration, opens the network programming interface, and promotes network innovation and network operation, the layered and opens interfaces provided by SDN, it also increases the network attack surface, resulting in many new security problems. In view of these various security problems in SDN, the scholars at home and abroad have made preliminary research and analysis on the vulnerability of controllers [2-3], the legality and consistency of flow rules [4-5], the vulnerability of Open Flow Protocol for south-facing interface [6-7], the security and standardization of north-facing interface [8-9]. Mahjabin et al., surveyed the distributed denial-of-service attack, prevention, and mitigation techniques [10] whereas, Wang et al., developed a software defined security networking mechanism (SDSNM) against DDoS attacks [11]. Fan et al, proposed an improved integrated prediction method of cyber security situation [12]. Liu et al, proposed monitoring DDoS by using SDN [13]. At present, most of the controlled environments in software definition networks are installed with simplified Linux operating system, which generally lacks reliable forensic security mechanism. Through self-adaptive audit mechanism [14], secure system management model and Linux system log record, the protection and evaluation of Linux kernel system were realized. This paper applies the physical memory forensic analysis method to the new network for the first time. Software-defined networking has positive significance for solving security problems under the new network architecture.

*Corresponding Author: Lijuan Xu; E-mail: xulj@sdas.org
DOI: 10.3966/160792642020052103027

The physical memory image file is a copy of the running memory data, when the host is running. It saves many key elements of information security event tracing and intrusion forensics analysis. By analyzing the memory image file, we can extract the memory information, network information, the running process information, loaded driver information, opened file information, registry and other volatile data, which are lost when the system shuts down. Therefore, it is of great significance to obtain the real-time information for reconstructing the scene of the case.

Since the network information exists in the physical memory image file, it can describe the communication situation of the computer with the outside world when it is being investigated, and can be used as an important evidence to judge whether the attacker is engaged in illegal network activities or not. Network attacks and crimes usually generate network connection information, such as IP address, the port number, etc. In computer online forensics, the network information can be obtained from the physical disk or through commands such as "tcpview" and "netstat", but the data obtained may change. Hence, we started with the physical memory image file to get real-time system information and network information from the host.

Network connection information resides in a physical memory as volatile data, and its acquisition depends on the correct physical memory analysis method. At present, there are many tools for analysing the information. Schuster et al., suggested a method to extract network information. In 2010, Ligh M et al., proposed another method to obtain the network information based on internal hash table. Ligh M et al., also presented a method for discovering malicious code hidden network connections through memory analysis. L Wang studied the method of extracting the network information from physical memory image files of Windows, Vista and Windows 7 systems [15-16]. The most advanced memory forensic tool is Volatility [17]. Volatility must rely on tedious configuration and it also depends on the system symbol of Linux kernel to make the profile file. The configuration method is cumbersome and has a great impact on the system from the perspective of forensics. The method (which is named SDN-NIMA) used in this paper acquires a volatile memory from Linux and Linux-based devices [18] which are analysed to find the network connection information, network configuration information, address resolution protocol information, and does not depend on the kernel symbol table, system version, and does not require complicated configuration steps. The calculations involved in the whole method are relatively less. The network information is quickly accessible with high acquisition efficiency, in addition it is also convenient to implement strongly through C++. The acquisition of network information through physical memory image files can be applied to the investigation and forensics. Thus, it is possible to check the information security threats, various computer network crime cases, and the abnormal behaviour detection under the new network architecture.

The outline of this paper is organized as follows. Section 1 introduces the security problems in the software definition networking and some important security solutions; Section 2, dwells on the development history and current situation of memory analysis, and intrduces important achievements of some scientists in memory analysis. The method of obtaining network information, including the method of obtaining network configuration information and obtaining address resolution protocol are elaborated in Section 3. Section 4 presents the comparative results obtained by the proposed method and Volatility forensic tools, and also give the advantages of the method. Section 5 summarises the content of the article and indicates the opportunities for future work in this field.

2 Related Work

Memory analysis started in 2002 as evinced by Kornblum's article published in DFRWS (Digital Forensic Research Workshop) on investigating volatile memory information for comprehensive and accurate access to cyber attacks and cybercrime information [19]. In 2005, a challenge game for the memory forensic analysis of the Windows system was organized (DFRWS Forensics Challenge, 2005), which mainly extracted the hidden process information contained in the physical memory dump file [20]. In 2006, A. Schuster proposed the process and thread search method in the Windows system image file [21]; Zhang et al. proposed a physical memory analysis method based on the KPCR (kernel processor control region) to locate process control block [22]. Whereas, Wang proposed a method for obtaining network connection information from 64-bit Windows operating systems [23]. Since then, the physical memory analysis was the prime area of research and development.

In comparison to the memory forensic analysis of Windows, the research work on Linux operating system is relatively less, but with the advent of new technologies such as cloud computing, big data, software-defined networking, etc., Linux forensics work becomes more and more important. The forensic analysis of Linux memory image files is even more important. A toolkit for analysing physical memory usage was developed from the Linux kernel [24]. DFRWS launched a memory forensic analysis challenge for Linux system [25]. In addition, the well-known Haker Conference (Black Hat, Def Con, Shmoo Con, etc.) started holding symposiums on memory forensic analysis from 2006. Gao proposed methods

for searching process according to init linear queue, hash table, process and queue, and further searched for other volatile information [26-27]. Zhang and others suggested the physical memory analysis method of Linux system based on kernel code reconstruction, but did not give the specific network information acquisition method [28]. This paper refers to the kernel data structure learning and the Linux kernel related books, and analyzes the physical memory image file for obtaining network configuration information and address resolution protocol.

This method can obtain network information correctly, including address resolution protocol, network configuration information and network connection information, which has been verified in the various kernel system versions. This method does not need to load complicated processes such as kernel files and system versions. It has less impact on the system, and its speed is relatively fast. It has positive effect and science for acquiring malicious behaviour in SDN environment.

3 Getting the Network Information

3.1 The Method for Obtaining Network Configuration Information

The structure layouts vary greatly depending upon the configuration parameters. For example, the layout of the net structure depends on the values of optional configuration parameters such as CONFIG_SYSCTL, CONFIG_IPV6, CONFIG_XFRM and CONFIG_ NETFILTER. Thus, to properly analyze a Linux image, the offsets of important structure members must be identified.

To obtain the configuration information of the network from the physical memory, we first need to find the starting virtual address of Dinit_net (Dinit_net is the default network namespace, which contains one or more network devices) in the kernel symbol table to obtain all the network card information. Each net structure has "dev_base_head" global variable which points to the "net_device structure" information. Secondly, we find the *ip_ptr global variable from net_device structure which points to the "in_device structure" information, and obtain information such as network interface, IP address, and network promiscuous mode from in_device structure.

3.1.1 The Relationship between Several Related Structures is Illustrated Below

The relationship among several structures for obtaining the network configure information is shown in Figure 1:

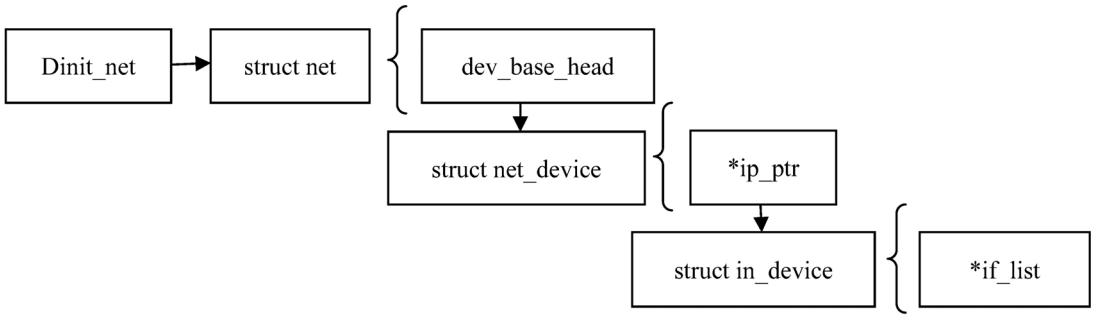

Figure 1. The relationship among several structures for obtaining the network configure information

3.1.2 Several Important Structures Information

The network namespace (net_namespace) in the kernel is represented by the struct net structure. In the Linux kernel, each network device (struct net_device) has its own network namespace. The net structure of the network namespace contains many fields, because in a network namespace, there may be many network devices, which are organized in the form of double linked list. The dev_base_head is the linked list head of the network device double-linked list; by default, in the Linux kernel, there will be a default network namespace named init_net as a global variable. All the network devices are organized by structure "list_head dev_base_list" in the form of double-linked lists.

The network abstraction layer uses the "net_device" structure to store information such as the device name, the mac address, the promiscuous mode, and IP address where, the pointer variable *ip_ptr points to the "in_device" structure.

The IP layer network device uses the in_device structure to store the IP address and ARP parameter information of the neighbour table. The structure and the offset of structural variables in memory are shown in Figure 2.

3.1.3 The Algorithm of Getting network Configuration Information

(1) Get the page directory address and virtual address of the system kernel symbol Dinit_net

Take the ubuntu kylin 14.04 system as an example. Firstly, use the liME tool, which is loadable kernel

```
struct net
 {
    atomic_t     passive;  +0x00
    atomic_t     count;  +0x04 /* To decided when the network*/
#ifdef NETNS_REFCNT_DEBUG
    atomic_t     use_count;
    spinlock_t     rules_mod_lock;
    struct list_head     list;     +0x0C    /* list of network namespaces */
    unsigned int     proc_inum;
    struct proc_dir_entry  *proc_net;
    struct proc_dir_entry  *proc_net_stat;
#ifdef CONFIG_SYSCTL
    struct ctl_table_set     sysctls;/* 36bytes */
#endif
    struct sock  *rtnl; /* rtnetlink socket */
    struct sock  *genl_sock;
    struct list_head     dev_base_head; +0X6C
.....
}
struct net_device
 {
       char       name[IFNAMSIZ]; +0X00
/* device name hash chain, please keep it close to name[] */
       struct hlist_node     name_hlist;
       unsigned long       state;
 struct list_headdev_list; +0X30
.....
       struct netdev_hw_addr_list    uc;        +0X10C /* Unicast mac addresses */
       struct netdev_hw_addr_list mc;/* ulticast mac addresses */
       bool       uc_promisc;
       unsigned int       promiscuity; +0X160
       unsigned int       allmulti;
       struct in_device __rcu          *ip_ptr; +0X174 /* IPv4 specific data*/
       .....
}
struct list_head list ;
struct list_head
{
 struct list_head *next, *prev;  0XF68fa800  0XF5A10830/*net structure*/
}
struct in_device
{
 struct net_device   *dev;
 struct in_ifaddr    *ifa_list; 0X0C //IP address chain
 .....
 struct neigh_parms  *arp_parms; /*neighbor table (arp) parameter configuration*/
  Struct ipv4_devconf cnf; /*IPv4 related device configuration*/
}
```

Figure 2. The structure and the offset of structural variables in memory

module to obtain the physical memory image file of the ubuntu kylin 14.04 system. Obtain the operating system version information and the virtual address of the system kernel variable swapper_pg_dir (0xC1A93000) by searching for "OSRELEASE=" in the image file. The value of swapper_pg_dir is the virtual address of the page directory in memory. If the operating system version information contains the "i686" character or the swapper_pg_dir value is 8-bit hexadecimal address, the system is a 32-bit operating system; if the operating system version information contains "x86_64" characters or the swapper_pg_dir variable value is a 16-bit hexadecimal address, the system is a 64-bit operating system, the base address of the system kernel variable swapper_pg_dir is the key to address translation. By searching the "vmcoreninfo" or "_text" strings, we can get all the system symbol tables' virtual addresses which include the Dinit_net structure's virtual address in memory.

(2) Address translation

According to the address of the page directory, which is "swapper_pg_dir", and the operating system version, you can get the address translation between the virtual address and the physical address. If the system version is 32_bit operating system, the PAGE_OFFSET is 0xC0000000 between the virtual address and the physical address; if the system is 64-bit, the PAGE_OFFSET is 0xffffffff80000000, for example if the virtual address of the page directory is 0xC1A93000, since the ubuntu kylin 14.04 system is 32_bit, the physical address of the page directory is 0x1A93000.

(3) According to the page directory address obtained in step 1 and the virtual address of the system kernel symbol Dinit_net, we can obtain the physical address of the Dinit_net symbol, and go through all the net structures in the memory image file according to the physical address of the Dinit_net, which is the default network namespace.

(4) Obtain the dev_base_head structure address at offset 0x6C by the physical address value of the net structure.

For each net structure, there may be multiple network devices in a network namespace, which are organized in a doubly linked list. Dev_base_head is a structure variable of "list_head" type in the net structure, which points to the header of the device's double-linked list of network devices.

Translate the virtual address of the net structure into the physical address and locate the address in the memory image file. Put 0x08 bytes from this position into a buffer. Read 4 bytes at buffer's offset 0x6c and judge whether the value is a pointer, which points to a net_device structure or not. If the value is a pointer which points to a net_device structure, go to step 5 otherwise, it indicates that the node's structure's address is zero and exits the procedure.

(5) Obtain the value of the dev_base_head variable in step 4, and obtain the starting address of the first net_device structure

Each net_device structure contains the network card in a promiscuous mode and MAC information. By analyzing the net_device structure, we can obtain MAC value information at offset 0x10C, and obtain the promiscuous mode of the network card at offset 0x160, this process is shown in Figure 3.

(6) Get the in_device structure according to the starting address "net_device" structure in step 5:

Since the *ip_ptr pointer is a kernel variable in the net_device structure, it points to the in_device structure which describes the device information in the IP layer. The address of the kernel variable ip_ptr pointer, that is, the starting address of the in_device structure, can be obtained at offset 0x174.

(7) Obtain the virtual address of the in_device structure in step 6, then obtain the virtual address of the pointer variable *if_list at the offset 0x0C, that is, the IP address list of the network card.

(8) Obtain the address of the next structure header, and judge whether the structure header is a dev_base_head structure, and if it is YES, then program will go to the step 3, otherwise go to the step 4. The flowchart is shown in Figure 4.

```
35A10800  6C 6F 00 00 00 00 00 00  00 00 00 00 00 00 00 00  lo..............
35A10810  00 00 00 00 60 16 A4 F5  00 00 00 00 00 00 00 00  ....`.¤õ........
35A10820  00 00 00 00 00 00 00 00  00 00 00 00 03 00 00 00  ................
35A10830  30 A8 8F F6 2C 78 98 C1  38 08 A1 F5 38 08 A1 F5  0¨ ö,x˜Á8.¡õ8.¡õ
35A10840  40 08 A1 F5 40 08 A1 F5  48 08 A1 F5 48 08 A1 F5  @.¡õ@.¡õH.¡õH.¡õ
35A10850  50 08 A1 F5 50 08 A1 F5  58 08 A1 F5 58 08 A1 F5  P.¡õP.¡õX.¡õX.¡õ
35A10860  60 08 A1 F5 60 08 A1 F5  68 08 A1 F5 68 08 A1 F5  `.¡õ`.¡õh.¡õh.¡õ
35A10870  69 7C 1B 00 05 00 00 00  00 48 1B 00 00 00 00 00  i|.......H......
35A10880  00 48 1B 00 00 00 00 00  20 00 00 00 00 00 00 00  .H...... .......
35A10890  01 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  ................
35A108A0  01 00 00 00 01 00 00 00  00 00 00 00 00 00 00 00  ................
35A108B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A108C0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A108D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A108E0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A108F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A10900  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A10910  A0 11 6C C1 A0 12 6C C1  00 00 00 00 40 17 6D C1   .lÁ .lÁ....@.mÁ
35A10920  09 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A10930  00 00 01 00 04 03 0E 00  00 00 00 00 00 00 00 00  ................
35A10940  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A10950  00 00 00 00 00 00 00 00  00 00 00 00 06 00 00 00  ................
35A10960  00 00 0A 0A 64 09 A1 F5  64 09 A1 F5 00 00 00 00  ....d.¡õd.¡õ....
35A10970  70 09 A1 F5 70 09 A1 F5  00 00 00 00 40 64 A2 F5  p.¡õp.¡õ....@d¢õ
35A10980  40 64 A2 F5 01 00 00 00  C0 73 A2 F5 00 00 00 00  @d¢õ....Às¢õ....
35A10990  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
35A109A0  00 00 00 00 00 B8 A9 F5  00 00 00 00 00 64 89 F6  .....¸©õ.....d‰ö
35A109B0  00 00 00 00 00 00 00 00  00 00 00 00 48 64 A2 F5  ............Hd¢õ
```

Figure 3. The offset of net_device structure in memory

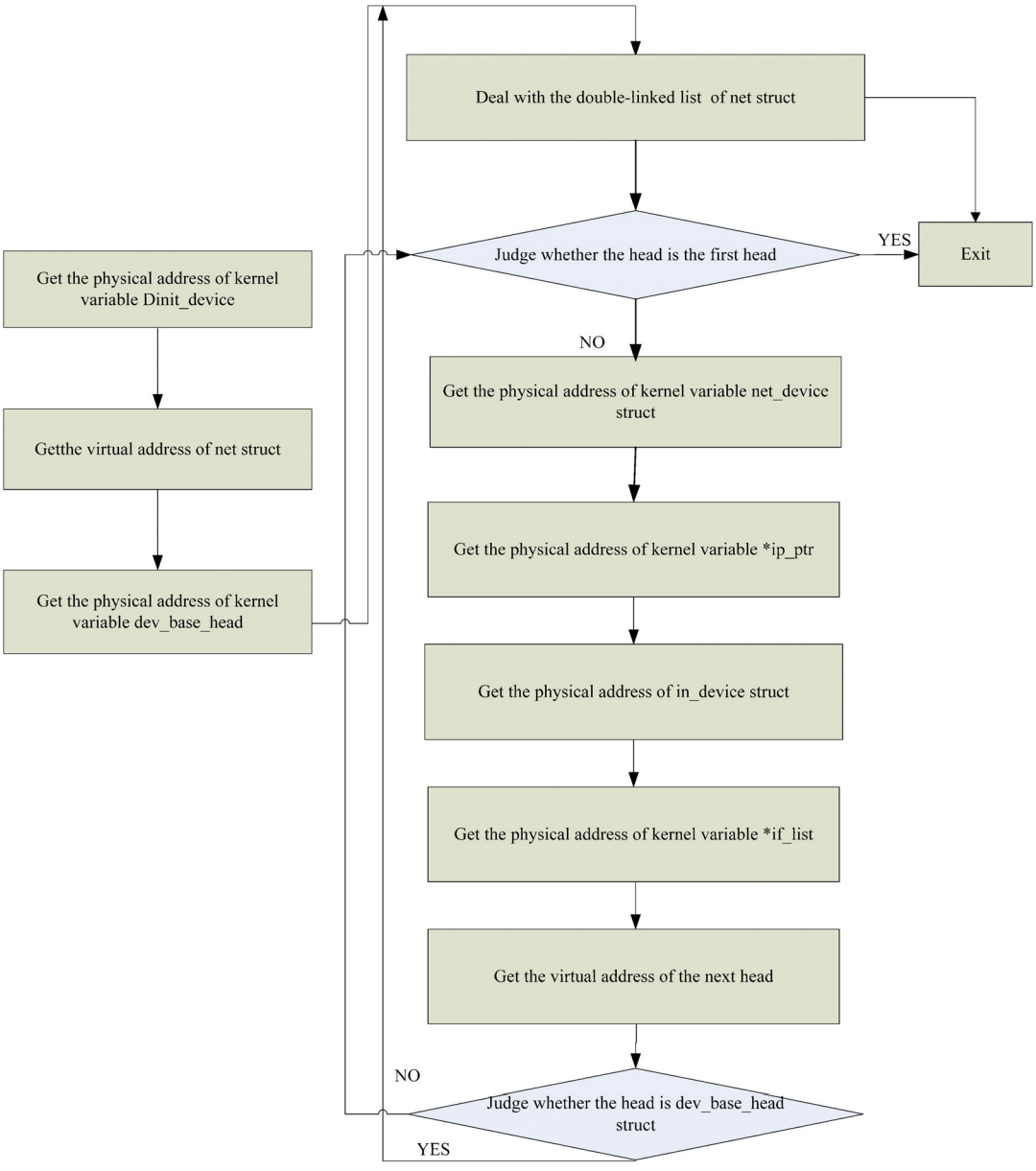

Figure 4. The flow of acquire network configure information

3.2 The Method of Obtaining the Address Resolution Protocol

The method of obtaining the address resolution protocol is similar to the method of obtaining the network configuration information. Taking the physical memory of the ubuntu kylin 14.04 system as an example, firstly, judge the operating system version, the page directory, and translate the virtual address and the physical address according to the system version and the page directory, and obtain all the system kernel symbol tables by searching the "vmcoreninfo" and "_text" strings. Locate the start address of the kernel symbol Dneigh_tables (0XC1B6F6BC), which is a pointer to neigh_table structure. By analyzing the structure of neigh_table, the starting addresses of the global structure variables viz. neigh_parms (0xF6A4B720), delayed_work, work_struct, timer_list and so on are found, and the starting addresses of the structure of neigh_hash_table are located. Because neigh_table is influenced by neigh_parms, delayed_ work, work_struct and other structures, the layout of these structures is influenced by the core variables such as CONFIG_LOCKDEP, CONFIG_TIMER_STATS and CONFIG_NET_NS. Since ubuntu kylin 14.04 is a 32-bit operating system, there is no lockdep_map structure defined, the int type is 8 bytes, the pointer type is 4 bytes, and unsigned long type is 4 bytes. The neigh_table structure occupies 0x148 bytes.

From the starting address of neigh_hash_table structure, locate the global variable ** hash_bucket structure (the starting address neighbour list in the hash bucket), hash_shift (the number of neighbours in the hash bucket), and obtain the starting addresses of each neighbour structure in the bucket. Finally, the MAC address, IP address and the device name of each neighbour item are obtained from the neighbour structure. Wherein, 31 four-byte positions in the neighbour structure are the MAC address, the first and second start of the last locations in the neighbour structure are the IP address and the device name. Then save the MAC address, IP address, device name to the file and html file. The relationship between several related structures is shown in Figure 5.

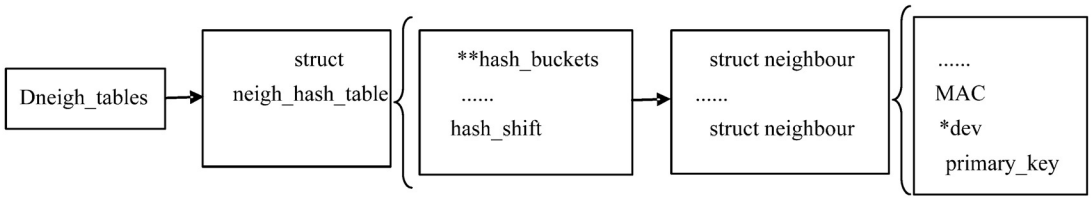

Figure 5. The relationships among several structures for obtaining the address resolution protocol

3.3 The Method of Obtaining the Network Connection Information

To obtain the network connection information, we need the Linux kernel structure including task_struct, files_struct, files, and other structures. From task_ struct->file->fd_array [fd], fd is the file descriptor. The file structure is found by fd which contains the directory item f_dentry. The f_dentry contains the inode, and the f_dentry can find the socket just by the inode structure, so the socket is found through the process descriptor. In the inode structure unsigned char i_sock variable is to record whether the inode is a socket, so you can find sockets, further, the network connection information is obtained.

4 Experiment

4.1 Compare the Results with Volatility Forensic Tool

We have done variable test across a wide range of operating system versions. In this paper, we use ubuntu kylin 14.04 as an example for verification testing. Firstly, install and configure the ubuntu kylin 14.04 environment in the virtual machine, and use "Lime" forensic tool to obtain the physical memory image file. The software part of the environment is as follows: In Linux 64-bit operating systems and 32-bit operating systems, the SDNProject command line program is developed separately using codeblocks 13.12 software. This program does not require the system.map system kernel file and Linux operating system version information, only the physical memory image file is required. Information such as network configuration information and network connection information and address resolution protocols can be analyzed. We store the information in the "configinfo_*.dat" and "arpinfo_*.dat" files respectively. Where, the "*" symbol represents the operating system version number. The network information obtained is shown in Figure 6 and Figure 7.

Install the ubuntu kylin 14.04 system, configure the volatility software, and create a profile file according to the usage method of volatility. The data obtained by using the volatility tool is shown in Figure 8 and Figure 9.

The obtained network connection information is shown in Figure 10.

```
[root@slave1 Debug]# cat ifconfiginfo_3.13.0-24-generic.dat
Interface   IP Address  MAC Address Promiscous Mode
lo [127.0.0.1]    at  00:00:00:00:00:00   FALSE
eth [172.16.8.191]    at  00:0c:29:aa:d4:b6   FALSE
```

Figure 6. The network configuration information obtained by memory analysis method

```
[root@slave1 Debug]# cat arpinfo_3.13.0-24-generic.dat
IP Address MAC Address Deive
[255.2.0.0]    at 33:33:ff:35:ea:f3   on eth0
[0.0.0.0]    at 00:00:00:00:00:00   on lo
[254.128.0.0]    at 00:00:00:00:00:00   on lo
[254.128.0.0]    at 00:00:00:00:00:00   on eth0
[127.0.1.1]    at 00:00:00:00:00:00   on lo
[224.0.0.22]    at 01:00:5e:00:00:16   on eth0
[172.16.8.254]    at 6c:50:4d:af:a3:c0   on eth0
[127.0.0.1]    at 00:00:00:00:00:00   on lo
[172.16.8.134]    at 78:2b:cb:88:ab:9c   on eth0
```

Figure 7. The address resolution protocol obtained by the memory analysis method

```
root@yangshm-virtual-machine:/usr/share/volatility# python vol.py -f /home/yangs
hm/ubuntu.lime --profile=Linuxubuntu1404x86 linux_ifconfig
Volatility Foundation Volatility Framework 2.6
Interface       IP Address          MAC Address        Promiscous Mode
--------------- ------------------- ------------------ --------------
lo              127.0.0.1           00:00:00:00:00:00  False
eth0            172.16.8.191        00:0c:29:aa:d4:ac  False
```

Figure 8. The network configuration information obtained by the volatility forensics tool

```
root@yangshm-virtual-machine:/usr/share/volatility# python vol.py -f /home/yangs
hm/ubuntu.lime --profile=Linuxubuntu1404x86 linux_arp
Volatility Foundation Volatility Framework 2.6
[ff02::2                                ] at 33:33:00:00:00:02    on eth0
[::1                                    ] at 00:00:00:00:00:00    on lo
[ff02::1:ffaa:d4ac                      ] at 33:33:ff:aa:d4:ac    on eth0
[ff02::fb                               ] at 33:33:00:00:00:fb    on eth0
[ff02::16                               ] at 33:33:00:00:00:16    on eth0
[172.16.8.246                           ] at 70:f9:6d:78:8c:f3    on eth0
[224.0.0.22                             ] at 01:00:5e:00:00:16    on eth0
[127.0.1.1                              ] at 00:00:00:00:00:00    on lo
[172.16.8.254                           ] at 6c:50:4d:af:a3:c0    on eth0
[172.16.8.117                           ] at c8:9c:dc:e4:81:58    on eth0
[127.0.0.1                              ] at 00:00:00:00:00:00    on lo
[172.16.8.96                            ] at 50:7b:9d:df:8a:25    on eth0
[224.0.0.251                            ] at 01:00:5e:00:00:fb    on eth0
[172.16.8.112                           ] at 44:39:c4:95:f6:a7    on eth0
```

Figure 9. The address Resolution Protocol obtained by volatility forensics tool

```
UNIX 7418                   init/1
UNIX 7700                   init/1
UNIX 8401                   init/1
UNIX 8559                   init/1
UNIX 8771                   init/1
UNIX 7691       upstart-udev-br/252
UNIX 7712         systemd-udevd/257   /run/udev/control
UNIX 7754         systemd-udevd/257
UNIX 7755         systemd-udevd/257
UNIX 8260           dbus-daemon/400   /var/run/dbus/system_bus_socket
UNIX 8383           dbus-daemon/400
UNIX 8384           dbus-daemon/400
UNIX 8402           dbus-daemon/400   /var/run/dbus/system_bus_socket
UNIX 8653           dbus-daemon/400   /var/run/dbus/system_bus_socket
UNIX 8733           dbus-daemon/400   /var/run/dbus/system_bus_socket
UNIX 8790           dbus-daemon/400   /var/run/dbus/system_bus_socket
UNIX 8969           dbus-daemon/400   /var/run/dbus/system_bus_socket
UNIX 8984           dbus-daemon/400   /var/run/dbus/system_bus_socket
```

Figure 10. The socket network information obtained by memory analysis method

The information obtained by the memory analysis method in this paper is compared with that obtained by the Volatility forensic tool, and the result of obtaining the network information is the same which verifies the correctness and reliability of this method. The time for obtaining the network information is compared through multiple comparisons. It is found that the proposed method is shorter than the Volatility by testing in different version systems, the result is shown in Figure 11. From the perspective of forensics, this method does not require cumbersome production steps and has less impact on the system.

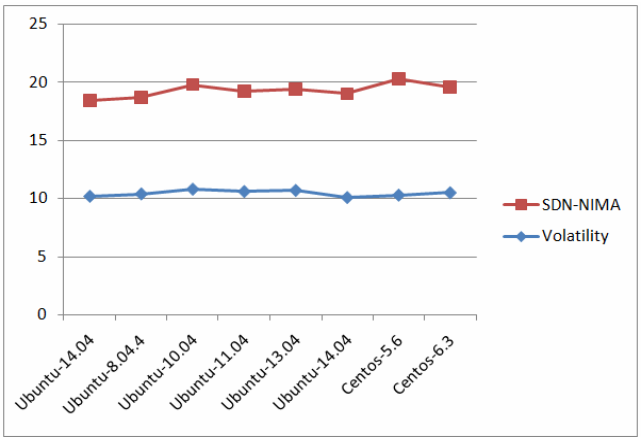

Figure 11. The execution time comparison by SDN-NIMA and Volatility

4.2 The Advantages of the Algorithm

The memory analysis method presented in the paper has general applicability, it does not depend on the kernel symbol table and system version, and does not require complicated configuration steps and provides a more general analysis method for Linux system memory analysis.

The calculations involved in the whole method are relatively less. The network information is quickly obtained, the acquisition efficiency is high, the method can also be easily implemented by C++ program, and the implementation is strong.

The physical memory stores the key information when the system is online. The method of obtaining network information from the volatile physical memory is an important task to solve the network security problem. It is of great significance to understand the mechanism of network attacks, respond quickly, and curb cybercrimes.

5 Conclusion

This article first introduced the security threats faced in SDN and related work on security issues, and gave the progress and problems of memory forensic. In view of the security problems in the SDN environment, we carried out research on the physical memory forensic analysis, and studied the characteristics of the Linux kernel object structure and the address translation under different versions. Finally, we proposed a method for extracting network information based on the Linux physical memory image file in the software-defined networking environment. The extraction method can obtain network information, including the network configuration information, the address resolution protocol, and the network connection information, which can be used to understand the network communication established between the target system and the external device, and can analyze whether the target system is attacked by the network or used to perform network attacks on other devices. By testing and verifying under ubuntukylin 14.04, the experimental results show that the physical memory analysis method adopted in this paper is correct, feasible, and the method can provide some clues for the security of the controller, southbound and northbound interfaces under the new network architecture. At the same time, we also give existing questions in the method. The next step is to expand the system version, solve the problem of the structures which depends on the system version, and build a sample library, strengthen the correlation analysis of online evidence, use the memory analysis technology and deep learning method to solve malicious code detection problems in the software-defined networking.

Acknowledgements

This work is supported by the Shandong Provincial Natural Science Foundation of China (Grant Nos. ZR2016YL011, ZR2016YL014), the National Natural Science Foundation of China (Grant Nos. 61702309), and the Shandong provincial Key Research and Development Program of China (Grant Nos. 2019JZZY020127, 2019JZZY020129, 2019JZZY010134, 2018CXGC0701, 2018GGX106005, 2017CXGC0701, and 2017CXGC0706).

References

[1] Open Networking Foundation, *Software-defined Networking: The New Norm for Networks*, ONF White Paper, April, 2012.

[2] P. Porras, S. Cheung, M. Fong, K. Skinner, V. Yegneswaran, Securing the Software-defined Network Control Layer, *2015 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, California, 2015, pp. 1-15.

[3] H. Wang, L. Xu, G. Gu, FloodGuard: A DoS Attack Prevention Extension in Software-defined Networks, *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Rio de Janeiro, Brazil, 2015, pp.1-12.

[4] B. Dai, H.-Y Wang, G. Xu, J. Yang, Opportunities and Threats Coexist in SDN Security, *Application Research of Computers*, Vol. 31, No. 8, pp. 2254-2262, August, 2014.

[5] W.-F. Xia, Y.-G. Wen, C.-H. Foh, D. Niyato, H. Xie, A Survey on Software-defined Networking, *IEEE Communications Surveys & Tutorials*, Vol. 17, No. 1, pp. 27-51, First Quarter, 2015.

[6] A. Lara, A. Kolasani, B. Ramamurthy, Network Innovation Using OpenFlow: A Survey, *IEEE Communications Surveys & Tutorials*, Vol. 16, No. 1, pp. 493-512, First Quarter, 2014.

[7] Open Networking Foundation, *OpenFlow Switch Specification* (Version 1.5.1), ONFTS-025, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-pecifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[8] X. Wen, Y. Chen, C. Hu, C. Shi, Y. Wang, Towards a Secure Controller Platform for Openflow Applications, *2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, Hong Kong, China, 2013, pp.171-172.

[9] J. Brazil, The Northbound API is the Key to OpenFlow's Success, https://www.sdxcentral.com/articles/contributed/the-northbound-api-is-the-key-to-openflows-success/2012/11/.

[10] T. Mahjabin, Y. Xiao, G. Sun, W. Jiang, A Survey of Distributed Denial-of-service Attack, Prevention, and Mitigation Techniques, *International Journal of Distributed Sensor Networks*, Vol. 13, No. 12, pp. 1-33, September, 2017.

[11] X.-L. Wang, M. Chen, C.-Y. Xing, Z. Sun, Q.-F. Wu, Software Defined Security Networking Mechanism Against DDoS Attacks, *Ruan Jian Xue Bao/Journal of Software*, Vol. 27, No. 12, pp. 3104-3119, December, 2016.

[12] Z. Fan, Z. Tan, C. Tan, X. Li, An Improved Integrated Prediction Method of Cyber Security Situation Based on Spatial-Time Analysis, *Journal of Internet Technology*, Vol. 19, No. 6, pp. 1789-1800, November, 2018.

[13] C.-H. Liu, Y.-T. Yeh, Monitoring DDoS by Using SDN, *Journal of Internet Technology*, Vol. 17, No. 2, pp. 341-348, March, 2016.

[14] L. Zeng, Y. Xiao, H. Chen, Auditing Overhead, Auditing Adaptation, and Benchmark Evaluation in Linux, *(Wiley Journal of) Security and Communication Networks*, Vol. 8, No. 18, pp. 3523-3534, December, 2015.

[15] J. Okolica, G. L. Peterson, Windows Operating Systems Agnostic Memory Analysis, *Digital Investigation*, Vol. 7, pp. 48-56, August, 2010.

[16] M. Ligh, S. Adair, B. Hartstein, M. Richard, *Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*, Wiley, 2010.

[17] Volatility: Linux Memory Forensics, https://code.google.com/archive/p/volaitility/vikis/Linux Memory Forensics.wiki.

[18] LiME~ Linux Memory Extractor, https://github.com/504ensicsLabs/LiME

[19] J. Kornblum, Preservation of Fragile Digital Evidence by First Responders, *2002 Digital Forensic Research Workshop*, Syracuse, New York, NY, USA, 2002, pp. 1-11.

[20] DFRWS 2005 Forensics Challenge, http://old.dfrws.org/2005/challenge/index.shtml.

[21] A. Schuster, Searching for Processes and Threads in Microsoft Windows Memory Dumps, *Digital Investigation*, Vol. 3, pp. 10-16, September, 2006.

[22] R.-C. Zhang, L.-H. Wang, S.-H. Zhang, Windows Memory Analysis Based on kpcr, *Fifth International Conference On Information Assurance and Security*, Xi' An, China, 2009, pp. 677-680.

[23] L.-H. Wang, L.-J. Xu, S.-H. Zhang, Network Connections Information Extraction of 64-bit Windows 7 Memory Images, *Forensics in Telecommunications, Information, and Multimedia*, Shanghai, China, 2010, pp. 90-98.

[24] P. Movall, W. Nelson, S. Wetzstein, Linux Physical Memory Analysis, *Proceedings of Freenix Track: 2005 Usenix Annual Technical Conference*, Anaheim, CA, USA, 2005, pp. 23-32.

[25] DFRWS 2008 Forensics Challenge, http://old.dfrws.org/2008/challenge/index.shtml.

[26] H. Gao, Y.-H. Gao, L.-Y. Zhou, Analysis of Linux Internal Access License Based on Process Organization, *China Science and Technology Expo*, No. 22, pp. 150-150, October, 2009.

[27] Y.-H. Gao, H. Gao, Y.-Q Ji, P. Bao, Analysis of Linux Internal access Credentials Based on Hash Table Lookup TCP Connection, *Silicon Valley*, No. 20, pp. 92, November, 2009.

[28] S.-H. Zhang, X.-X. Meng, L.-H. Wang, An Adaptive Approach for Linux Memory Analysis Based on Kernel Code Reconstruction, *EURASIP Journal on Information Security*, Vol. 2016, No. 1, pp. 1-13, December, 2016.

Biographies

Shumian Yang is currently an Associate Research Fellow with the Shandong Computer Science Center (National Supercomputer Center in Jinan), China. She received a master's degree in computer science and technology from Shandong Normal University, in 2007. Her main research interests include computer forensics, memory forensics and network security.

Lianhai Wang is currently a Research Fellow at Shandong Computer Science Center (National Super Computer Center in Jinan), China. He received Ph.D. degree in computer science and technology from Shandong University, China. His research interests include information security and computer forensic and block chain.

Shuhui Zhang is now an Associate Research Fellow at the Shandong Computer Science Center (National Supercomputer Center in Jinan), She received Ph.D. degree from the Department of Computer Science and Technology at Shandong University of China in 2019. Her current research interests include

computer forensics, cloud forensics, blockchain, and network security etc.

Dawei Zhao is currently an Associate Research Fellow with the Shandong Computer Science Center (National Supercomputer Center in Jinan), China. He received Ph.D. degree in cryptology from the Beijing University of Posts and Telecommunications in 2014. His main research interests include network security, complex network, and epidemic spreading dynamics.

Lijuan Xu is currently an Associate Research Fellow with the Shandong Computer Science Center (National Supercomputer Center in Jinan), China. She received a master's degree in computer science and technology from Shandong University, in 2007. Her main research include network security, industrial internet security and computer forensics.