Digital evidence in virtual honeynets based on operating system level virtualization

Pavol Sokol, Peter Pisarčík

pavol.sokol@upjs.sk, peter.pisarcik@upjs.sk

Department of Computer Science
Faculty of Science
University of Pavol Jozef Šafárik in Košice
Košice, Slovakia

Abstract

The network forensic analysis is the branch of the digital forensic analysis. It focuses on capturing and analyzing network traffic in order to identify network threats and attackers and subsequently to obtain legal evidence. There are lots of methods suggested for network forensic analysis, including honeypots and honeynets. Both are common elements to achieve goals of digital forensic analysis. In this paper we focus on virtual honeynets based on operating system-level virtualization. As a representative of this type of virtualization, we choose, in order to develop virtual honeynets, an open-source virtualization platform – OpenVZ and FreeBSD jail. Digital evidence is any type of documentation, which satisfies the requirements of evidence in proceedings, and digital means that exists just in electronic form. This paper offers a discussion about digital evidence, sources of digital evidence, as well as its most important requirements, what is necessary for the fulfillment of the validity of digital evidence. Due to these requirements, we consider integration of several sensors into the discussed type of virtual honeynet. Our implementation of honeynet hence includes sensors in memory (includes buffers, caches, I/O ports etc.), CPU, harddisk drives and network devices. Using these sensors, the digital evidence is stored at the safe place. Using OpenVZ and FreeBSD jail technologies enables a user to access all memory blocks, mainly the processes, and to identify honeypots (virtual machines, containers), on which the processes are running. Thank to this information we are able to detect all changes in processes after intruder penetrated the server. Actually, within CPU-based digital evidence we use direct access to CPU from hypervisor kernel. We are able to read all registers and all instructions received from and sent to the computer processor. This advantage provides malware intrusion detection and analysis. Within the storage-based digital evidence we have direct access to the file system and we are able to identify all changes in file system, such as file creation and its modification. In this paper we discuss the advantages of OpenVZ and FreeBSD jail virtualization platform due to the implementation of the above-mentioned sensors. Since in this type of virtualization all virtual environments (honeypots) share one kernel of operating system, it is sufficient to implement the sensors in one place only - in the kernel. Moreover, we describe the high effectiveness of such data collection in this paper, as well as we discuss the safe conditions of integral data collection, which is necessary to obtain valid digital evidence.

Keywords: digital evidence, virtual honeynet, operating system virtualization, sensors, OpenVZ

1 Introduction

In the present, our everyday lives are becoming more and more dependent on information technologies. Due to this fact, the network and information security are areas that require more attention and improvement. Traditionally, these areas are primarily intended to be defensive. When the protection of the information system is broken and the attacker controls the system, it is necessary to use the tools and

methods of the network forensic. Network forensics can be defined as the use of scientifically proved techniques to collect, fuse, identify, examine, correlate, analyse, and document digital evidence from multiple, actively processing and transmitting digital sources for the purpose of uncovering facts related to the planned intent, or measured success of unauthorized activities meant to disrupt, corrupt, and or compromise system components as well as providing information to assist in response to or recovery from these activities [1] . There are lots of proposed models for network forensic that consist of many different phases. One of them is honeypot framework, which is explained in detail bellow.

This paper is organized as follows: In section 2 the theoretical background of honeypots, honeynets, virtualization and virtual honeynets is examined. Section 3 presents related works to virtual honeynets and virtualization. In section 4 we provide an overview of OpenVZ and FreeBSD jail concept and approaches to their implementation. In section 5 we propose virtual honeynets sensors. In section 6 we outline digital evidence obtained by proposed sensors. Section 7 offers conclusion of this paper, as well as our opinion on the future of these sensors.

2 Theoretical background

In this section, we provide a reader with basic theoretical background to virtual honeynets. The definition, advantages, disadvantages, implementations, and categorization of this framework are given below.

2.1 Honeypots and honeynets

As mentioned above, honeypot presents a model for network forensic. It deals with collecting information about intruders, malware, boots and other threats in order to learn the details concerning intruders' behaviours, practices and aims. Lance Spitzner defines Honeypots as an information system resource whose value lies in unauthorized or illicit use of that resource [2]. A honeypot can be also defined as a computing resource, whose value is in being attacked. A honeypot computer system is a system that has been deployed on a network for the purpose of logging and studying attacks on the honeypot. These systems may be made purposely insecure in order to lure attackers to study their techniques, tools, and motivations [3].

Using of honeypots has several advantages and disadvantages. According to I. Mokubeand M. Adams [4], there are various advantages of the honeypots, such as:

- **Small data sets** - on to the traffic that comes to honeypots;
- **Minimal resources** – capturing of the attacks require minimal resources;
- **Simplicity** – honeypots can be developed, updated and maintained as simple;
- **Discovery of new tools and tactics** - honeypots can discover new attackers' methods and tools.

Same authors also describe some of the disadvantages of the honeypots. These are [4]:

- **Limited vision** – honeypot only captures network traffics when the attacker directly interacts with them;
- **Discovery and fingerprinting** – honeypots can be identified as the trap;
- **Risk of takeover** - the honeypot may be used to attack other systems.

The honeypots can be divided into two main types, based on their purpose and level of interaction. According to purpose, there are the two types of honeypots, namely research honeypots and production honeypots. **The research honeypot** is designed to gain information about blackhat community and it does not add any direct value to the organization which has to protect its information [5].

The production honeypot is used within an organization's environment to protect the organization and help mitigate the risk [5]. An example of the production honeypot is a honeypot that captures, collects, and analyses malware for anti-virus, or intrusion detection system signatures etc.

Second classification is based on level of interaction. We know low-interaction and high-interaction honeypots. The difference between these classes is the extent to which the attacker is allowed to interact with the system. **Low-interaction honeypots** implement targets to attract or detect attackers by using software to emulate the characteristics of a particular operating system and network services on a host operating system. The goal of a **high-interaction honeypot** is to give the attacker access to a real operating system where nothing is emulated or restricted [2]. High-interaction honeypots are useful in situations where one wishes to capture details of vulnerabilities or exploits that are not yet known to the public.

One type of honeypot is called honeynet. Specifically, it is a high-interaction honeypot designed to capture extensive information on threats [6]. Also, it is primarily used to learn about the tools, tactics, and motives of the blackhat community. Its purpose is to collect the attackers' information. It is done by creating a network of systems to be attacked. Nothing is emulated due to fact that it uses real systems and applications [7]. Honeynet provides real operating systems to attackers to interact with [6].

Honeynet has evolved across three generations. **Honeynet generation I** was the first honeynet that was developed in 1999. This honeynet has simple methodology, limited capability, and it is highly effective in detecting the automated attacks. It runs at ISO/OSI layer 3 and uses the reverse firewall for data control. **Honeynet generation II** presents more complex honeynets to deploy and maintain. This honeynet, which was introduced in 2001, ran on ISO/OSI layer 2. It examined the outbound data and blocks, passed or modified data. **Honeynet generation III** was released at the end of 2004. This type of honeynet shares the same architecture with its predecessor. The differences between the two showed up as improvements in deployment and management.

There are several key requirements that must be implemented into honeynet [8]:

- **Data Control** is the first requirement whose purpose is to control the attacker´s activity.
- **Data Capture** monitors and logs all of the attacker's activities within the honeynet.
- **Data Collection** - is the case that organization has more than one honeynet, all data has to be captured and stored in one central location.
- **Data Analysis** is an ability to analyse the data that is being collected from the honeynet.

There are four issues associated with the use of honeynets [6]:

- **Harm** - attacking another system used by honeynet.
- **Detection** - when the honeynet is identified, information about attackers' methods and behaviours is reduced due to ignoring or bypassing the honeynet.
- **Disabling honeynet functionality** - attacker is able to disable data control or data capture capabilities.
- **Violation** - this point is related to the first point. Attackers may attempt cybercriminal activity using the honeynet to attack another system.

2.2 Virtualization and virtual honeynets

Technology, which allows multiple virtual machines to run on a single physical machine, is called the virtualization [8]. This technology divides the resources of a computer into multiple separated Virtual Environments (VEs). According to Amit Singh, the virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments by applying one or more concepts or technologies, such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others [9].

There are different levels of virtualization such as full virtualization, paravirtualization and operating system (OS) level virtualization.

The concept of the all virtualization method consists of three parts **guest operating system, hypervisor and host operating system (optional)**.

Guest is the virtualized operating system utilized within any of the below virtualization strategies. Therefore, a guest can refer to a virtual private system (in operating system-level virtualization), or a virtual machine (in full virtualization or paravirtualization). A system that controls the host processor and resources is called **hypervisor** (virtual machine monitor). **Host operating system** is the actual operating system on which the virtualization takes place.

In this paper we focus on **operating system level virtualization**. In this method of virtualization, the kernel of the operating system allows for multiple isolated system instances. These instances are known as containers (Virtual Private Servers or Virtual Environments). Advantages of this method are: usually minimal or no overhead, high performance, dynamic resource allocation and best scalability. On the other hand, disadvantages include single (same) kernel per physical server and the fact that guest and host operating system must have the same type of kernel. So, in host operating system based on the UNIX kernel, the operating system based on windows kernel cannot be run. Examples of operating system-level virtualization are OpenVZ, Solaris Zones, Linux-VServer and FreeBSD Jail.

Above we defined honeynet and virtualization. According to these definitions, **virtual honeynet** can be defined as a complete honeynet, running on a single computer in virtual environment [8]. Virtual honeynet is a technology that virtually implements many different operating systems in one hardware computer, and hence instead of having a honeynet of different physically separate honeypots, all the honeypots will be virtually set in one machine and still appears to the attacker as different separate machines [10].

The use of the virtual honeypots has several advantages and disadvantages. The main **advantage** is the fact that instead of using some physical servers to create the honeynet, it is possible to create it with only one machine. It is also easy to be managed, since all the honeypots are configured in one machine [10]. On the other hand, the **disadvantage** would be that the types of services that can be monitored by a virtual honeynet are limited by the hardware and virtualization software. For example, UML technology and OpenVZ technology are only supported in Linux-based operating systems. To be more specific, if the attacker is able to compromise the hypervisor and control it, he has the control over all virtual systems - honeypots.

The Honeynet Project defines two types of virtual honeynets, namely self-contained and hybrid [6]. **Self-contained virtual honeynet** is contained to one physical system. This type of virtual honeynet typically consists of a gateway device with firewall for data control and data capture and honeypots within the honeynet. Main advantages of the self-contained virtual honeynet are portability and cost effectivity. Virtual honeynet can be placed anywhere in the network and it can be ready to capture and collect network traffic during the virtual honeynet is connected. Self-contained virtual honeynet has some drawbacks, such as high rate of failure, requirement of powerful hardware, specialized software and security. The combination of a classic honeynet and a virtualization method is presented by **hybrid**

virtual honeynet. Data capture and data control are run on an isolated system. All honeypots are virtually run on a single machine. The main advantages of this approach are flexibility and security. The main disadvantage of this type of virtual honeynet lies in the fact that complete honeynet is based on one single physical machine.

2.3 Digital evidence

Since main object of this paper is digital evidence, it is necessary to define evidence, digital evidence and network-based evidence, and to outline the challenges relating to this type of evidence. **Evidence** can be defined as information or signs indicating whether a belief or proposition is true or valid. Evidence is also information used to establish facts in a legal investigation or admissible as testimony in a law court [11]. Any documentation which satisfies the requirements of evidence in a proceeding, but which exists in electronic digital form, is defined as **digital evidence** [12]. **Digital evidence** of an incident is digital data that contain reliable information that support or refute a hypothesis about the incident being investigated. An object is evidence because it played a role in an event that supports or refutes an investigation hypothesis [13]. **Network-based evidence** is digital evidence that is produced as a result of communications over a network. An example of it includes logs, network traffic, browser activity etc. [12].

In **network-based evidence,** there are some challenges in several areas [12]:

- **Acquisition -** it can be difficult to locate specific evidence in a network environment. Networks contain many possible sources of evidence, for example log servers, access points.
- **Content -** file systems are designed to contain all the contents of files and their metadata. Network devices often have very limited storage capacity.
- **Storage -** network devices commonly do not employ secondary or persistent storage.
- **Privacy -** there may be legal issues, involving personal privacy that are unique to network-based acquisition techniques.
- **Seizure -** seizing a hard drive can inconvenience an individual or organization. A clone of the original can be constructed and deployed so that critical operations can continue with limited disruption.
- **Admissibility -** file system-based evidence is now routinely admitted in both, criminal and civil proceedings.

When collecting and analysing evidence, there is a general four-step procedure containing [14]:

- **Identification of evidence** – it is necessary to distinguish between digital evidence and junk data. For this purpose, the investigator has to know, what data is collected, where it is located and how it is stored.
- **Preservation of evidence** – the digital evidence must be preserved as close as possible to its original state. Any changes made during this phase must be documented and justified.
- **Analysis of evidence** – the stored digital evidence must be analysed to identify the relevant information and creation the chain of events.
- **Presentation of evidence** - it is important part. Presentation of digital evidence must be understandable.

In the following text, we focus on identification and preservation of digital evidence only.

3 Related works

In our research we focus on the combination of two areas, namely **virtual honeynets** and **operating system-level virtualization**. Research in area of the virtual honeynets has resulted in a number of papers, discussing specific topics, concerning virtual honeynets based on full virtualization and paravirtualization, but no operating system-level virtualization. A large amount of helpful information about virtual honeynet, based on **full virtualization,** can be found in the Experiences with III generation Virtual Honeynet [8]. Example of virtual honeynet, based on **paravirtualization implementantion,** is NoSE. In [15] Stumpf et all developed a system called Network Simulation Environment (NoSE) to simulate arbitrary network environments on a single Linux machine. NoSE integrates different virtual machine emulators, such as Xen, User-Mode-Linux and QEMU, the Linux kernel's bridging facilities, and various network management and monitoring tools. User-Mode Linux as virtualization technology to setup a virtual honeynet can be found in [16]. Kwong Yan in his paper proposed new architecture of virtual honeynet that is different to the GenII honeynet and utilizes Security Enhanced Linux to isolate the untcusted honeypots from the completely trusted honeywall. Another interesting project of virtual honeynet can be found in Design of virtual honeynet collaboration system in existing security research networks [17]. This paper describes the virtual honeynet collaboration system based on XEN virtualization technology.

In research in **field of operating system-level virtualization**, there is a very interesting research in [18]. This paper contains a large amount of helpful information about virtualization. In this article authors select three typical open source virtual machine monitors (i.e. OpenVZ, Xen and KVM) as the delegates of operating system-level virtualization, para-virtualization and full-virtualization to evaluate their performance with a combinative method that measures their macro-performance and micro-performance as a black box and analyzes their performance characteristic as a white box. In [19] authors presented the challenges for the implementation of a system-level virtualization solution for high performance computing and they identified six domains for which a research effort is or has to be initiated.

4 Honeynet based on operating system virtualization

In our research we conduct research in field of operating system-level virtualization using OpenVZ virtualization platform and we also start with FreeBSD jail technology. The first reason for selecting OpenVZ as primary Honeynet platform is for us performance capabilities of OpenVZ. According to [20], it is comparable with virtualization alternatives, as full virtualization (KVM), as well as paravirtualisation (XEN). In this article, OpenVZ clearly leads to tests focused on disk operations and it is also a leader in system calls by switching the CPU context. In remaining performance areas, OpenVZ is fully compensated for both mentioned virtualization technologies.

4.1 Specification of OpenVZ

OpenVZ (Open VirtualiZation) is a software virtualization for Linux kernels. Because the host operating system, as well as virtualized systems, shares one system kernel, we refer to it as to an operating system-level virtualization, which means virtualization method, where the kernel of an operating system allows for multiple isolated user-space instances instead of just one. Each of the instances is called Virtual Environment (VE) or Virtual Private System (VPS), and it looks like a real server, from the point of view of its users.

There are currently several softwares that enable operating system-level virtualization but OpenVZ achieves the highest level of isolation. OpenVZ provides full separation of container file system from the entire host file system, like the standard system call chroot(), but it completely removes all the security

risks, associated with using chroot system call. OpenVZ fully supports disk quotas, respectively two-level disk quota: each container can have its own disk quotas.

Each virtual system has its own tree of processes, which appears (inside the container) to be completely independent of other containers processes. OpenVZ thus separates and isolates the processes memory space of each container. However, the host system shows all processes of all containers and there is a possibility to interact with them as if they were in a single operating system environment. OpenVZ also contains the full isolation of IPC objects such as shared memory, semaphores, and messages.

The OpenVZ also provides isolation of network traffic. The implementation includes two types of network interfaces that can be used inside the container as network devices: Veth (Virtual Ethernet Device) and Venet (Virtual network device). Veth virtual interface operates on the second layer ISO/OSI model, Venet operates on the third layer. The OpenVZ developers' website offers the following description:

Main advantage of the **veth** devices lies in the fact that they can be bridged together and/or with other network devices. The veth device allows broadcasts in the virtual private system which makes it possible to use a DHCP server or a samba server inside the virtual private server. This device is usually bridged directly to the host physical ethernet device. It is the same as the real ethernet device on a standalone machine. Users in virtual private server can access a veth device as a real ethernet interface. On the other hand, the root user is the only one that is privileged to access the veth device.

Second network device is a **venet**. It is a bit faster and more efficient than a veth device. This device has also some disadvantages. Administrator of the host operating system can only assign an IP in the virtual private system. In the veth device, the network settings can be fully done on the virtual private system side by the administrator of the virtual private system. On the other hand, in the veth network device, the administrator of the virtual private system setup IP address masks the address, the gateway address, etc. In spite of the fact that both devices are ready for the IPv6 addresses, an IPv6 address is generated only from MAC [18].

OpenVZ also implements **complete isolation of the physical components** of the machine. Hence, each particular container can allocate hardware, such as serial or parallel port, disk partition and so on. This is related to the existence of a fair multi-level I/O scheduler, which distributes the available I/O bandwidth.

OpenVZ offers **two-level fair CPU scheduler**. On the first level, the OpenVZ scheduler decides, which container to choose, based on per-container cpu-units values. On the second level, the standard Linux scheduler decides which process from selected container to run, using standard Linux process priorities.

Security of OpenVZ technology is based on the security model (deny by default), tracking mainstream security fixes, stable kernel without new bugs, security review by Solar Designer and activity monitors. At present, OpenVZ has over 400.000 usages and the wide community helps with its security.

4.2 Specification of FreeBSD jail

In the FreeBSD operating system exists as in other UNIX system kernel system call chroot() - this allows the change of the root of directory structure to the subdirectory, which provides a safety benefit, the file system isolation. Because there are lots of known techniques, how to bypass chroot() system call and get access to the entire directory structure, chroot() is not appropriate for large applications. FreeBSD therefore implements a tool for software virtualization at the operating system, FreeBSD JAIL. This security tool is more effective as the system call chroot(), and is resistant to vulnerabilities that characterised chroot() system call . Unlike the Linux kernel, which uses a variety of patches (as has been described above) - OpenVZ technology, the kernel of FreeBSD has JAIL technology as an integral part: jail(2) as system call and jail(8) as utility developed for jail management.

Jail, at first, isolates processes and processes groups by changing the root filesystem in subdirectory structure that is similar to chroot() system call. In addition, however, it completely isolates processes and processes groups from processes outside the jail. After starting the jail, it is not possible to add any process inside the jail that is not a child of an already running process in jail - no process can leave the jail and no process can interact with processes outside the jail. Jail is also started with an exact IP address; all services are forced to use that particular IP address and this IP address cannot be changed inside jail. Also, root, the superadministrator, cannot create any new system devices. Among others, restrictions in jail include: the inability to change the running kernel and to load kernel modules; inside the jail is also no way to change network settings, connect and disconnect filesystems, changing some sockets, modify sysctl and so on.

Jail uses the concept of client (jail) - server (host). Changes in the system performed on the host, in the main system, are realized also on the jail. Changes performed in jail are valid only for this particular jail.

4.3 OpenVZ and FreeBSD jail in process of data collection

Based on the described properties of operating system-level virtualization in OpenVZ and FreeBSD jail implementation, we describe their main advantage as being useful tools for forensic analysis. Here are the **main advantages** of OpenVZ, in comparison to the FreeBSD jail, in terms of building useful honeynets:

- **Direct access to file system of container from the host** file system. In both virtualization platforms, virtualized environments file systems are part of host system directory structure, so any changes in container file system can be regularly detected from host system. At the time of running the container, we can appropriately interact with file system of container to achieve some operations. This is very useful in detecting and preventing the intrusion without detection from container side of view. It should be noted that FreeBSD distribution contains implementation of ZFS file system, which, in connection with FreeBSD jail virtualization, supports Copy-On-Write, which greatly speeds up disk operations.
- **Using virtual network interface** inside the container. By using a virtualized interface venet or veth in OpenVZ technology, it is possible to achieve a complete separation of the network data flow of containers. On the other side, it is possible to create specialized configuration that redirects the network data flow from (or to) a container in a prepared network traffic analyzer. In FreeBSD jail, there is direct implementation of network isolation in FreeBSD kernel, so OpenVZ brings more flexibility. Although the latest version of FreeBSD (9.0) implements new "vnet" virtual interface, which provides considerable network flexibility for FreeBSD jail: full virtualized networking inside jail, jails have their own loopback interface, it is possible to use advanced network features inside jail, e.g.: IPSec. However, both platforms, because they use the same kernel for all containers, are a point of interest, where to integrate network sensor.
- **All I/O operations pass through a one** system kernel. This feature is typical only for OpenVZ implementation because FreeBSD does not implement I/O rate limiting. In our research, we focused on creating sensors to capture data flow from the peripheral device, especially from a keyboard. Creation of sensors needs the Linux kernel modification.
- **Access to all processes that are running on a physical machine**. Since both virtualization platforms share one kernel that is used in the creation of each process, there is possibility of monitoring all running processes in one place, namely in the host system. The advantage of the Linux operating system is that it offers all the information about running processes via a virtual file system, a PROCFS. Unlike the FreeBSD operating system, PROCFS is an optional component, and hence must not be used. But eventually, PROCFS can be mounted with full

compatibility to the Linux PROCFS. This makes it possible to read all processes' details of all containers from host system. PROCFS file system will be described in the next chapter.

- **Direct access to the instructions of the CPU and memory**. Again, due to the fact that all virtual containers share one operating system kernel, it is possible (with direct editing of the kernel source code) to capture all data of processor and memory undetectably.

5 Implemented sensors in virtual honeynet based on OpenVZ

5.1 Network sensors

In our research in field of sensors of the virtual honeynet, based on OpenVZ, we focus on the implementation of network sensors that collect network data incoming from and outgoing to a honeynet core interface. We divide this implementation into four phases according to types of network sensors. The main reason for the development of up to four sensors is the necessity of selecting the most effective method for use in honeynet. These are the proposed sensor types:

- **First kernel module.** The main aim of this kernel module is to create a virtual network interface and to create I/O entry point in a virtual file system PROCFS in order to collect the captured data. The data are captured by the bridge, which interconnects the external network interface with the developed virtual network interface. Implementation of this sensor was already published in international conference proceedings.
- **Second kernel module.** Goal of the second kernel module, whose implementation is the subject of this paper, is also to create a virtual network interface but without creating I/O entry point in a virtual file system PROCFS. In contrast with the proposal of previous sensor type, captured data are stored in virtual file system. Hence, this proposal of kernel module assumed development of specialized file system. By this kernel module, the data are captured also by the bridge which interconnects the external network interface and a developed virtual network interface. Implementation of this sensor was already sent in international conference proceedings.
- **Integration** of the capture routine directly into the **network card driver.** Due to decrease latency, we assume to integrate the third type of sensor directly into source code of network card driver. The decision, where to store the collected data, depends on results of previous sensor implementations, either in PROCFS, or in own virtual file system.
- **Integration** of the capture routine directly into **the kernel source code**. Similarly to the third type of implementation, in order to decrease latency, we suggest integration of the third type of sensor directly into source code of kernel source code. The decision, where to store collected data, again, depends on results of first and second sensor implementations, which means either in PROCFS or in own virtual file system.

5.2 Proc file system

Proc file system (PROCFS) is a virtual file system in UNIX-like operating systems, especially in GNU/Linux. Its original goal was to provide information about running processes but during a short period of evolution, many important functions were added. It caused that PROCFS became environment for communication with the kernel, what includes management and bidirectional control of the system. Data structure of PROCFS consists of one directory for each running process, several directories for access to the kernel parameters and some special files with relevant information. Each process directory is named

by process PID and consists of sub-directories and files, which offers detailed information about the process, associated with the process directory.

5.3 Proposed virtual file system

The main goal of the Linux file system is a behaviour, which appears to be a canonical tree, but it is often created by multiple file systems. This is possible by the existence of Virtual File System (VFS) in the Linux kernel. Virtual file system is an abstract layer, allowing working with files in a uniform manner, regardless of their allegiance to a particular file system.

The driver of file system, the kernel module, is similar to the character device driver, but it must also implement special features for mounting and unmounting file systems and functions to work with the directory structure. Many features usually need not to be implemented, since there is a system library "libfs", which consists of several generic functions' implementation. Functions of the virtual file system can be divided into the following categories: functions of filesystem type, superblock functions, functions for i-node, file operation functions, functions of the directory tree space, and functions of cache directory entries.

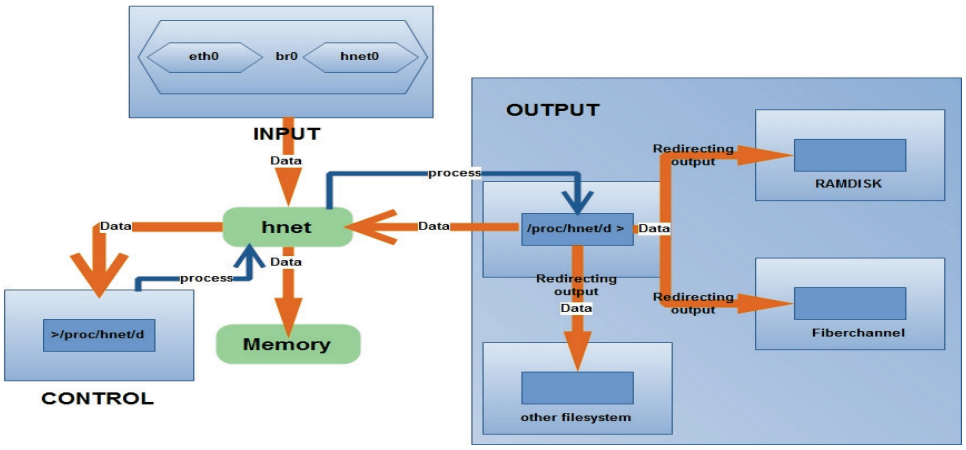

Figure 1: Algorithm of data flow organized by proposed kernel module.

5.4 Description of the implementation of network sensors

At this point we describe the first and the second phase of implementation, what means implementation of first and second sensor type. Both implementations include a creation of dynamic kernel module. The main aim of kernel module is to create the virtual network interface, hnet, and I/O entry point in PROCFS and creation of own filesystem, which is visible at Fig.1. Both sensor types share the same first step – virtual network interface creation.

In order to use network interface in operating system, the phase of creation of the interface is in kernel driver. This is done by network driver, when kernel system call ndo_open arrives. Linux kernel calls ndo_open, for example, when the command "ifconfig up" is typed. Our implementation of the kernel module creates specialized virtual ethernet network interface, hnetX, where X is index of the interface. Network interface driver normally implements two basic functions, namely send data and receive data. If the user process calls function, for example, "send()", the data passes through kernel network layer and comes into the network interface driver. At this point, the kernel sends into the network driver ndo_start_xmit, what, from the point of view of driver, means receiving and sending data. Our

implementation includes only one type of the basic network driver functions, namely data receiving. And received data are in our network driver neither processed, nor sent anywhere.

Since we require access to data that enter and leave the honeynet, we decided to use the network bridge in the first sensor type. Main aim of the network bridge is to link the real external honeynet interface and proposed virtual interface. Using this interconnection, our virtual network driver is able to access all the data that are captured by real honeynet network interface,

At this point, the first step is finished. The second step of the implementation is for each type of sensor different. In first type of sensor the captured data are stored in a special queue, which is dynamically allocated in the kernel memory, when the kernel module is loading into the kernel. Size of the queue is given by the module parameter (the integer value) and it specifies the number of network frames that can be maximally stored in it. Every packet is indexed in queue.

The access to stored network data, captured by the first type of sensor and saved in a special queue, is provided via I/O entry point at the root of the directory structure in a virtual file system PROCFS. This entry point consists of one directory called "honeynet", which contains one regular file, named "data". Data capturing is carried out simply by reading of the file "data". Reading of the file "data" is destructive. It means that the data that were successfully read from the queue are considered outdated. This activity is carried out by moving the queue pointer, from actual position on the first index, which was not read yet. If the end of the queue is reached, the queue pointer returns to the beginning of the queue.

Input function of the I/O entry point in PROCFS, which brings control options for proposed virtual network interface, is realized by writing an integer value into the file "data". Valid input is value between 0 and n, where number 0 means the complete deletion of the queue. On the other side, value, greater than zero and at the most equal to n, determines number of records stored in queue to be read. Value n is identical to the value, parameter, entered in stage of loading our kernel module. This parameter determines the maximum size of the queue, especially number of queue entries.

In second type of sensor the captured data are stored in a structure of proposed and implemented file system: data captured by the virtual network device hnet are stored in dynamically allocated structures in kernel memory. New memory block, a new element in the array of structure, is allocated every hour. Size of the memory, which is allocated, is determined by the parameter that is specified when the kernel module is loading in the operating system. Each element of the array is presented as a regular file in self implemented file system. If the allocated memory would not suffice, the memory would be reallocated.

In terms of features implemented in the creation of the proposed file system is to be noted that there were used many of generic functions. Own functions have been implemented only in order to read the directory structure and to read the file contents. Reading of captured data is carried out by the simple reading of the regular files in created file system. Reading of the data is destructive. It means that the data, successfully read from file, are removed from memory. This activity is carried out by freeing allocated memory and by removing file from directory structure.

6 Conclusion and future works

This paper has provided a brief overview of virtual honeynets based on operating system-level virtualization, mainly based on OpenVZ and FreeBSD Jail. We have also shown, that the operating system-level virtualization in comparison with other virtualization technologies has provided some advantages for network forensic framework - honeynet such as direct access to shared memory and possibility to interact with physical machine CPU, the specialized network devices, the direct access to file system of container from the host file system and access to complete processes tree.

In paper we focused on description of OpenVZ and FreeBSD Jail and provided their comparison due to usage in process of data collection. We proposed four types of sensors for digital evidence capturing and

collecting. In paper we described the implementation of first and second type of sensor. Implementation of first sensors consists of virtual network driver which captures data in kernel space memory and entry point in virtual file system (ProcFS) via which the data is available for data analyser. In second type of sensor we also implements virtual network driver for data capturing, but data is available via proposed virtual file system.

In future the research will continue with implementation of third and fourth type of sensor. Challenges for future research are proposal and implementation of other sensor such as memory sensor, file system sensor and process dispatcher sensor and are integration of sensor based on other operating system level virtualization technologies.

Acknowledgement

Research of digital evidence in virtual honeynets based on operating system level virtualization is supported by grant **VEGA 1/0479/12.**

References

[1] Palmer G.: A Road Map for Digital Forensic Research, in Digital Forensic Research Workshop, Utica, 2001. pp. 15-30.

[2] Spitzner, L.: Honeypots: Tracking Hackers, Addison Wesley, 2002, pp. 1-430.

[3] Spitzner, L.: The Honeynet Project: Trapping the Hackers, IEEE Security & Privacy, March/April 2004, pp. 15-23.

[4] Mairh, A., Barik, D., Verma, K. and Jena, D.: Honeypot in network security: a survey, In Proceedings of the 2011 International Conference on Communication, Computing & Security (ICCCS '11). ACM, pp. 600-605.

[5] The Honeynet project: Know Your Enemy: Learning about Security Threats (2nd Edition), Honeynet Alliance, 2004, p. 768

[6] Chang J.: Design of virtual honeynet collaboration system in existing security research networks, Communications and Information Technologies (ISCIT), 2010 International Symposium on , vol., no., 26-29 Oct. 2010, pp.798-803

[7] Mokube, I. and Adams, M.: Honeypots: Concepts, Approaches, and Challenges, ACMSE 2007, March 23-24, 2007, pp.321-325.

[8] Abbasi F. H. and Harris R.J.: Experiences with a Generation III Virtual Honeynet, Telecommunication Networks and Applications Conference (ATNAC), 2009 Australasian , pp.1-6, 10-12

[9] Singh, A.: An Introduction to Virtualization, 2004, www.kernelthread.com/publications/virtualization

[10] http://www.honeynet.org

[11] Oxford Dictionaries Online – English Dictionary and Language Reference, http://oxforddictionaries.com

[12] Davidoff, S. and Ham, J.: Network Forensics: Tracking Hackers through Cyberspace, Pearson, 2012.

[13] Carrier, B. D. and Spafford, E. H.: Defining Event Reconstruction of a Digital Crime Scene. Journal of Forensic Sciences, 49(6), 2004

[14] Vacca, J. R.: Computer Forensics: Computer Crime Scene Investigation, Charles River Media, Inc., Hingham, 2002, p. 731

[15] Stumpf, F., Görlach, A., Homann, F., Bruuckner, L.: NoSE - building virtual honeynets made easy, In Proceedings of the 12th International Linux System Technology Conference, Hamburg, Germany, 2005.

[16] Kwong, L. Y.: Virtual Honeynets Revisited, In Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE, pages 232-239, IEEE, 2005

[17] Márquez F.G. and Cambronero D.F.: Use of VNUML in Virtual Honeynets Deployment, IX Reunion Espanola sobre Criptologia y Seguridad de la Informacion (RECSI), Barcelona (Spain), September 2006

[18] www.openvz.org

[19] Vallee, G., Naughton,T., Engelmann,C., Ong,H. and Scott,S. L.: System-level virtualization for high performance computing, in PDP '08: Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), 2008, pp. 636–643.

[20] Jianhua Ch. et al.: Performance Combinative Evaluation of Typical Virtual Machine Monitors in Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 96-101