Honeypot-Architectures using VMI Techniques

Stefan Floeren
Betreuer: Nadine Herold, Stephan Posselt
Seminar Future Internet SS2013
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: floeren@in.tum.de

ABSTRACT
Honeypots are an effective tool to gain information about sophisticated attacks and zero-day exploits. With rising popularity of virtual machines in the World Wide Web, systems using virtual honeypots also get more interesting. After giving an introduction into traditional honeypot systems, this paper first describes *VMScope*, a VMI-IDS (virtual-machine-introspection-based intrusion detection system) which focuses on providing a tamper-resistant but thorough honeypot surveillance system. Then *Collapsar* is described, a system of multiple virtual honeypots that logically resides in different networks with the purpose of detecting attacks that span across multiple networks. Finally, a combination of both systems is proposed and the capabilities are discussed.

Keywords
Honeypot, VMScope, Collapsar, VMI IDS

1. INTRODUCTION
Recently, Internet users have been under attack from many threats such as viruses, worms, and Trojan horses. But lately the number of reports about attacks of single hackers or whole groups on specific targets, for example Sony or Facebook, has increased drastically. It is therefore desirable to detect these attacks and at best prevent them in the first place. To achieve this, it is essential to gather detailed data about the course of actions of an attacker. Honeypots of different types have emerged as a viable source for this kind of information. However, as honeypots get more and more popular, it is increasingly interesting for attackers to detect, avoid or disable them to make sure their methods remain hidden.

Another development in the World Wide Web is the conversion from physical servers to virtual machines. This is noticeable in the increasing number of offers of virtual server offers by most hosting providers, for example the *Elastic Compute Cloud (EC2)* from Amazon Web Services. Following this development, honeypots running in virtual machines are also getting more popular.

This paper gives an introduction to two honeypot systems, *VMScope* and *Collapsar*. Therefore as a necessary prerequisite an overview of intrusion detection systems and how they work is given. The paper then introduces honeypots and how they can be categorized depending on their interactivity and behavior. The next section introduces *VMScope*, a honeypot system that uses virtual machine introspection. Then, *Collapsar*—a system to provide a whole honeypot farm—is discussed. Finally, both systems are compared and a suggestion on how they could be combined is presented.

2. INTRUSION DETECTION SYSTEMS
An *Intrusion Detection System* (IDS) is a system that monitors computers or networks for suspicious activity or traffic and it is a necessary part of honeypot surveillance. It can detect known attack signatures, for example buffer overflows, port scans, and operating system fingerprinting[8]. IDSs can be separated into different categories which are described in the following paragraphs.

2.1 Network-based
The first type is the *network-based IDS* (NIDS). It captures all traffic from and to a monitored host. As capturing takes place outside of the monitored system, a NIDS is invisible to an attacker. Therefore, captured data is trustable even if the host is corrupted. However, no internal activities of the host can be captured. It is also highly ineffective if the traffic is encrypted because no information but flow data can be gathered out of this kind of traffic[6]. Common software used to gather this kind of information includes Tcpdump[16] and Wireshark[19].

2.2 Host-based
The second kind of IDS is the *host-based IDS* (HIDS). Using this approach, the IDS is integrated into the host as a part of the operating system—for example as a kernel module—or running as an application. Because it is running on the host itself, the IDS can collect internal data, for example system calls. This data can be used to generate a detailed analysis of the course of actions of an attacker. It is also possible to check file integrity and log files automatically. The drawback of this method is, that after an intrusion an attacker can detect the HIDS and tamper with the logging module which makes all further log files after the intrusion worthless[3].

2.3 Virtual-Machine-Introspection-based
The last category is the *Virtual-Machine-Introspection-based IDS* (VMI IDS). This approach is a special kind of a host-based IDS which tries to mitigate its weaknesses.

2.3.1 General Approach
The idea of a VMI IDS is to move the surveillance part outside of the host by preserving the thoroughness of a host-based IDS. This is achieved by virtualizing the surveilled

doi: 10.2313/NET-2013-08-1_03

host and putting the IDS outside of it, for example into the virtual machine monitor. The *virtual machine monitor* (VMM) is the software which provides the virtualization. The machine the VMM runs on is called *host*, the operating system running inside a VMM is called *guest*. The VMM runs directly on the host's hardware and virtualizes it to transparently multiplex its resources to one or more VMs. It needs to perform this abstraction in a way that a software that would run on the original hardware will also run on the virtualized one. VMMs can run as a simple operating system directly on the hardware (for example Xen[20]) or as a user process running on the host's operating system[3], like VMware[18]. If the VMM runs as software on the host machine, it is necessary to convert system calls from the guest system so that the host operating system understands them, especially if the guest and host OS differ. This process is called *binary translation*[6].

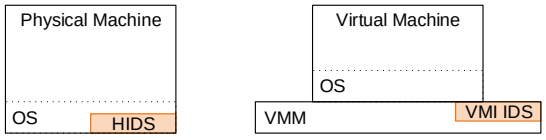

Figure 1: Comparison between a HIDS and a VMI-IDS. Based on Fig. 1 of [6]

Because the VMM keeps control of the hardware itself, it has a complete view of the guest's system state, including CPU and I/O registers, volatile memory and stable storage. With this information, a detailed observation and analysis of the guest system is possible. This method is called *Virtual Machine Introspection* (VMI)[10]. An IDS using VMI to surveil a host is called *VMI IDS*.

Additionally to the advantage of invisible and detailed data collection, VMMs offer further capabilities that can help in analyzing an attack. As the state of a VM is completely visible from the VMM, it is possible to save the current state of the virtual machine. By comparing such a snapshot of a known good state with a corrupted system, it is possible to perform deep forensics of corrupted files or memory areas off-line or to replay an attack step by step[3].

2.3.2 Virtual Machine Monitor properties

In order to be useful for VMI purposes, a VMM needs to fulfill some properties[3]:

Isolation: A software running inside of a VM cannot change anything outside the VM itself, regardless of its internal execution level. This ensures that the VMM and therefore the IDS cannot be tampered with even if an attacker has complete control over the VM.

Inspection: In order to achieve the same logging capabilities as a host-based IDS, the VMM needs to access all states of a virtual machine: CPU states and registers, whole memory and all I/O device states and content. Therefore, it is very difficult for an attacker to evade an IDS inside the VMM.

Interposition: On specific operations inside of the VM, like privileged instructions, the VMM needs to interrupt because of its design. Because of this, a VMI IDS can hook into these instructions for logging purposes with only minimal modifications to the code of the VMM.

2.3.3 Challenges

On integrating the logging functionality into the VMM some design trade-offs should be considered.

The first challenge is the **integration** of the IDS. As the VMM needs to be modified to add the logging behavior, the source code of the virtualization software needs to be available. Therefore proprietary software like VMware can only be supported with the help of the developing company. A VMM's source code is relatively small, but as its correct behavior is critical for the security of the host system, it must be tested and validated thoroughly[3]. On integrating the logging functionality into the VMM it is important to consider how much code is integrated, because it should provide the same level of reliability as the VMM itself. It therefore should be the goal to integrate as less code as necessary. This needs to be traded off with the wish to add as much functionality as possible[3].

The next challenge to solve is **performance**. Virtualization always has an overhead in comparison to a hardware PC because all privileged calls or hardware accesses need to be trapped by the VMM, modified and then sent to the host. The introduction of additional logging into the VMM further expands this overhead; the introduced increase depends on how many and which system events should be monitored. While some features—like accessing hardware states—normally do not cost any performance, trapping interrupts or memory access can affect the performance in a quite worse way because of their frequency of occurrence[3].

Another challenge is **VM environment detection**. It is possible to detect if a program is running inside of a virtual machine or on a physical environment and there already exists malware that shows different behavior inside a VM. Although there are opportunities to prevent some VM detection methods from succeeding, this is not strictly necessary, as virtualization gains popularity[6] and is not the exception any longer in the World Wide Web (for example virtual root servers).

The last challenge is the **semantic gap**. The virtual machine monitor and therefore the IDS sees the guest environment in a strictly binary form. To generate useful log files or check the integrity of a specific file on the guest's file system, the intrusion detection system needs to know the on-disk structure or the memory management of the guest's operating system. To provide services that rely on this higher-level information, the lost information has to be reconstructed in some way. To gain full semantic information, all the guest system's abstractions need to be reimplemented. As these abstractions are specific for each OS, they cannot be reused and therefore this approach does not scale. However, some basic abstractions are shared across many operating systems. This includes virtual address spaces, network protocols and file system formats. Therefore those generic abstractions can be implemented and reused for many operating systems[1, 10].

3. HONEYPOTS

A *honeypot* is a system inside of a network which is used to detect previously unknown attacks and vulnerabilities. Its sole purpose is to get broken into and every connection to it is considered suspicious[11]. Honeypots generally can be divided into three categories depending on the level of interaction they support[7].

3.1 Honeypot Types

High-interaction: First, there are *high-interaction honeypots*. They allow an attacker to control a whole system with almost no restrictions. Of course it is necessary to prevent an attacker from infecting other computers than the honeypot, therefore some kind of network filtering or controlling is needed. These honeypots are most effective in detecting new vulnerabilities, but they also introduce high risks and need to be supervised tightly.

Medium-interaction: The next category is the *medium-interaction honeypot*. It imposes more restrictions to the system than a high-interaction honeypot and therefore may not gather as much information, but is is also less risky than a high-interaction honeypot[7]. For example *chroot* can be used to set up this kind of honeypots.

Low-interaction: The last category is the *low-interaction honeypot*. Here, only some part of an operating system is simulated, like the network stack[11]. This provides the lowest risk in trade off for much less detection capabilities. They are also the easiest ones to set up[7].

3.2 Honeypot Tools

In this section the standard tools for high-interaction and low-interaction honeypots, Sebek and Honeyd, are presented.

3.2.1 Sebek

Sebek[13] is the de-facto standard tool for monitoring high-interaction honeypots. It is a host-based intrusion detection system and installs itself as a kernel module. It wraps a number of system calls to its own implementations that record context information—for example the process ID of the calling process—and the arguments before calling the kernel function. The captured data is then sent to a remote server which logs the received information.

3.2.2 Honeyd

Honeyd[4] is a low-interaction honeypot tool. It runs as a daemon and simulates the TCP/IP stack of a target operating system, supporting TCP, UDP and ICMP. It listens for packets destined for the configured virtual hosts. It is also capable of simulating whole network topologies with routers, end hosts and link characteristics (latency, packet loss). Its goal is to fool tools like Nmap[9] or traceroute to believe in the presented network topology and to fake the fingerprints of the simulated machines. It therefore adjusts fields in the TCP, UDP and IP header and the closed port behavior to match the operating system's which should be simulated[11].

3.2.3 Comparison

Honeyd is effective in detecting network scans as the daemon can manage whole virtual networks. However, as it only implements the network stack, it is easily detectable by trying to use default services that run on most machines, like ssh on Linux, as these protocols are not implemented in honeyd[8].

Sebek can detect attacks on the host it is running on. It is capable of gathering detailed information about the course of actions until it is detected. After detection, captured data cannot be trusted anymore because the attacker can tamper with it. Detection of Sebek can be done with relatively little effort as described in [2].

4. VMSCOPE

The first honeypot monitoring system described in this paper is VMScope. It is a VMI-based intrusion detection system.

4.1 Description

VMScope's purpose is to replace host-based IDSs like Sebek. Its main goal is to provide logging functionality that is more resistant to attackers while keeping the same level of detail of logging. Therefore, it should not be possible to tamper with the generated log even if the attacker has full control over the attacked PC. To achieve this, VMScope captures not just some specific system calls like Sebek but all. This is necessary because it is possible to substitute different calls with each other and therefore avoid detection if only some system calls are surveilled[2]. Additionally, it is capable of capturing all system events from the first moment of booting while host-based IDS cannot start capturing until they are started—and maybe already circumvented.[6].

4.2 Implementation

This section describes the details of the proof-of-concept implementation, based on QEMU[12], a software-based virtual machine monitor, described in [6].

4.2.1 System calls

To capture all system calls, one of QEMU's key features, binary translation, gets extended. VMScope wraps the QEMU handling of system events, like interrupts, so that before (*pre-syscall*) and after (*post-syscall*) executing the system call a VMScope function is called (Figure 2). The callback function before executing the system call collects context information, the callback behind tracks the return value. To correctly interpret the gathered information, the exact calling conventions from the system that invoked the system call needs to be known. It is especially important to know where parameters are passed to the system call and where the return value is stored—on Linux for example, this is mainly done via registers. The numeric parameters are interpreted with run-time information to identify the semantic meaning. Therefore it is possible to log the call with detailed information like the path given to *sys_open* instead of just printing the content of the register which contains a virtual memory address.

4.2.2 Challenge - Multitasking

In a single task environment, it is sufficient to use a variable to pass data from the pre-syscall function to the post-syscall function. However with multitasking it is possible that two concurrent system calls occur and in between those two calls a context switch occurs. It is therefore necessary to keep track of the state and lifetime of a process running inside

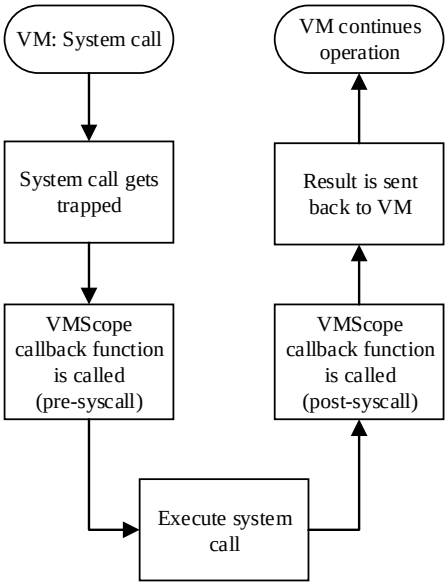

Figure 2: System call handling of VMScope

the VM. If the VM is running Linux it would be possible to include this information inside the VM in the *thread_union* struct which contains information about each thread. This would made tracking of process lifetime inside the VMM unnecessary and accessing the data from the post-syscall function would be very efficient, although this would introduce a possibility to tamper with the data from inside the virtual machine. Therefore, the current VMScope implementation maintains this information inside the VMM itself.

4.3 Evaluation

In [6] the efficiency and effectiveness of VMScope has been evaluated.

4.3.1 Detected attacks

First, some tests with real world attacks were performed.

The first test was a comparison of VMScope with Sebek on a system which was affected by NoSEBrEaK, a tool to circumvent Sebek[2]. After activating NoSEBrEaK an encrypted network connection was initiated, a rootkit was installed and two shell commands were executed. While Sebek could not detect any actions, VMScope could capture and interpret all commands. The VMScope log file, starting directly at the boot of the system, also shows how Sebek is hiding itself as *iptables-nat.ko*.

The second task was a test with a Slapper worm with well-known behavior. The log files generated by VMScope shows that the infection occurs in three steps: (1) Exploit a buffer overflow to gain a remote shell. (2) Upload and compile the worm source code. (3) Launch the newly generated binary and start a new round of infection. These steps could be verified by an internal system call tracker and a Slapper worm analysis tool.

A third test was the actual usage of VMScope honeypots to monitor real-world attacks. One of this attacks, an OpenSSL vulnerability exploit in Apache, was described further. Through the VMScope log files, the behavior of the attacker could be investigated in detail: After opening a remote console through a specially prepared HTTP request, a *ptrace* vulnerability was exploited to gain root privileges. Then, two rootkits, a ssh daemon and an IRC bot, got installed. With the help of the log files generated by VMScope, every keystroke could be reconstructed.

4.3.2 Performance

To evaluate the relevant performance, some system tasks (compressing files with *gzip* and compiling the Linux kernel with *make*) and some benchmarks (*ApacheBench*, *Nbench* and *Unixbench*) were done on an unmodified QEMU-VM and a VM with VMScope running. Those benchmarks were selected to give a good overview of the whole system's performance on different common tasks. The different benchmarks spaced from 96.4 % of the base system's performance for Apache to 85.6 % for Unixbench. The results indicate that the overall performance loss is at most 15 % and therefore the loss is reasonable.

4.3.3 Limitations

As the VMScope code runs inside the VMM, it needs to be as trustworthy as the VMM's code itself. This is essential to prevent compromisation of the VMM or the host computer.

To interpret system call events, knowledge of system calls and their calling conventions is essential. It might be possible to remap those system calls in a non-standard way and therefore mislead or corrupt VMScope. In how far this is possible depends on the VMM itself, as security-enhanced ones could detect and prevent such remapping. Also, accurate identification of dynamic kernel objects remains a challenge. [6]

5. COLLAPSAR

The second system described in this paper is Collapsar. It is based on high-interaction honeypots running in virtual machines, although it does not use VMI. Physically, all virtual machines reside in the same computing center, logically, they are distributed between different networks. The goal of Collapsar is to provide centralized management, convenient data mining and attack correlation[7].

5.1 Architecture

Collapsar consists of three main parts: the *redirector*, the *front-end* and the *honeypot* itself (Figure 3). It also contains *assurance modules* to analyze and control the actions of an attacker[7].

5.1.1 Redirector

The redirector is located inside of the network the honeypot pretends to be part of. It captures all traffic which is designated for the honeypot, filters those packets to strip out sensitive information, depending on policies imposed by the administrator of the network, and finally encapsulates and dispatches them to the Collapsar Center.

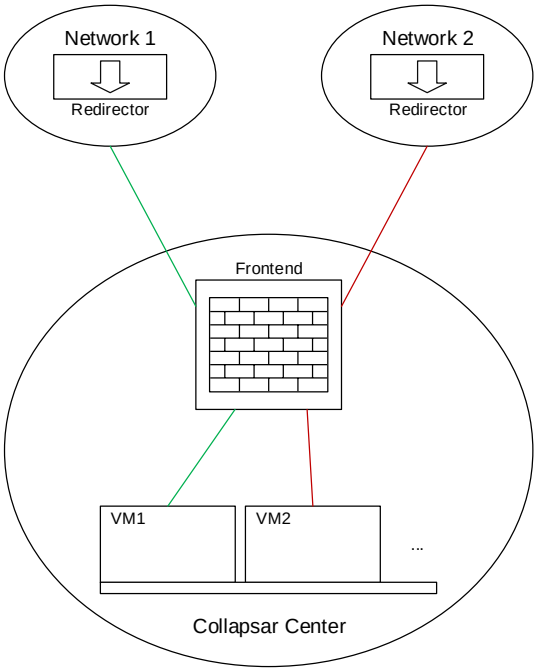

Figure 3: Architecture of the Collapsar network

5.1.2 Front-end

The front-end is the gateway of the Collapsar Center. It receives and processes the packets it receives from the redirectors and dispatches them to the corresponding virtual machine. It also ensures that packets from the honeypots are sent to the right redirector to make communication with the outside possible. It also provides functions of a firewall to prevent an infection from spreading through a honeypot.

5.1.3 Virtual honeypot

The virtual honeypot is a virtual machine running inside the Collapsar Center. Although the traffic is redirected, an attacker should think he is talking directly to a machine inside the corresponding logical network.

5.1.4 Assurance modules

Assurance modules provide necessary function for attack analysis and minimization of risks. There are several default modules. First, there is the *logging module* which collects data through sensors inside of the VM. This data is stored outside the VM to prevent tampering with the log files. Second, the *tarpitting module* is used to mitigate risks from the honeypot. It throttles outgoing traffic and cripples packets matching a known attack signature. The last module is the *correlation module*. It compares the log of the different honeypots and therefore can detect network-based attacks like network scans or the propagation of a worm inside the network.

5.2 Implementation

This section describes the Collapsar implementation as presented in [7].

5.2.1 Traffic redirection

Traffic redirection is the process of redirecting the packets from the network the honeypot logically resides into the Collapsar Center. It can be implemented in two different ways: router-based and end-system-based. In the router-based approach a router inside of or at the border of the network gets configured to tunnel the traffic to the Collapsar front-end. The advantage of this method is that it is highly efficient. On the other hand it requires to configure a router which may conflict with policies for network administration. In the end-system-based approach an application on one host inside the network redirects the traffic to the front-end. This approach results in easier deployment of the whole system. In the proof-of-concept implementation the application level redirector itself is a virtual machine. This allows further packet filtering regarding policies and rewriting of packets before forwarding. It can also be extended easily to provide functions of a network-based intrusion detection system.

5.2.2 Traffic dispatching

Traffic dispatching happens in the front-end and describes the process of inserting the packets received from the redirectors into the correct virtual machine. In fact the front-end is similar to a transparent firewall. If the front-end is also implemented as a virtual machine, it can be used as another point of logging. Ideally, packets should get forwarded directly to the intended honeypot. To support multiple VM systems, the packets are injected into an internal Collapsar network and from there they are picked by the honeypots. The advantage of this method is that no changes to the VM provider's code have to be done and therefore proprietary solutions can be used. But this method induces additional overhead and it causes *cross-talk*. This means that two honeypots which reside in two different logical networks get the packets destined for the other honeypot on the same internal network which can be exploited by an attacker to detect that he is inside the Collapsar network. A solution for this problem is to directly inject the packets into the VM's network interface. This eliminates the problem of cross-talk but it requires to modify the source code of the VM provider. Therefore, this method cannot be implemented on proprietary software, like VMware, without much effort.

5.2.3 Virtual honeypot

The honeypots used by Collapsar are virtual machines, the current implementation supports either VMware or User-Mode Linux (UML)[17]. Depending on the virtualization solution, either direct injection (UML) or an internal network (VMware; *vmnet*) is used.

5.2.4 Assurance Modules

Assurance modules are deployed in different locations. Logging modules like network sniffers are deployed in both redirectors and front-ends. Inside of the honeypot, sensors to capture all actions are deployed. For network sniffers, Tcpdump[16] and Snort[14] are used; inside the virtual machine Sebek[13] or a kernelized snort are deployed.

Tarpitting modules are deployed in the front-end and the redirectors. In the presented implementation, snort-inline[15], a open source modification of snort which accepts packets from iptables instead of libpcap, is used. Its purpose

doi: 10.2313/NET-2013-08-1_03

is to rate limit out-going connections and defuse malicious packets matching known attack signatures.

5.3 Evaluation

In [7], Collapsar has been evaluated in regards to real world incidents and performance.

5.3.1 Detected attacks

To show the effectiveness of Collapsar, a testbed containing five production networks with honeypots running different operating systems has been used. Several attacks are described in [7]. They show the detection capability of attacks against single honeypots. As this functionality is provided by Sebek this is the expected behavior. An interesting fact is, that one of the attacks the exploit originated from a different IP address than the connection to the installed SSH server. This could be a hint for the usage of a stepping stone, a machine already controlled by an attacker to perform further attacks. One attack, that shows the capabilities of Collapsar is the detection of a whole network scan originating from one IP address, as honeypots in different logical networks received the same ICMP packet.

5.3.2 Performance

The performance has been tested in respect to two aspects in regards to the main bottle neck of Collapsar—the network performance. First, the overhead of the virtualization alone and second the overhead of the whole Collapsar System including traffic redirection is measured.

The impact of virtualization was compared between UML, VMware and an unvirtualized machine. In case of ICMP performance, tested with different payload sizes, both virtualization mechanisms show similar latency degradation. However in TCP throughput, tested by transmitting a 100 MB file with different buffer sizes, VMware shows almost no difference to the unvirtualized machine. UML in contrast has a worse performance than VMware. The reason for this is the different virtualization techniques. VMware implements binary rewriting, which uses breakpoints to catch sensitive instructions but executes the other instructions faster. UML virtualizes all system calls and therefore fully emulates a x86 machine.

With the whole Collapsar system in place, both tests were repeated. The latency for ICMP packets raised significantly for both virtualization systems. This is caused by the Collapsar design and usage of a end-system-based redirector. While this is detectable for a nearby attacker, remote attackers may not be able to distinguish this delay from normal delay through the internet. This performance could be improved by using router-based redirection or hardware-based virtualization. In contrast to the results without the traffic redirector, UML now has better performance than VMware in both TCP throughput and ICMP latency. This could be caused by the direct traffic injection when using UML.

5.3.3 Limitations

Although Collapsar is very powerful in detecting attacks it has some limitations. First, to really be able to benefit from the honeyfarm and its cross-network detection abilities, it is necessary to build a very large network. As described in [7], 40 virtual machines are far too less. A bigger project to gather information from over 2000 sensors exists with the name Internet Storm Center[5], but it is not capable of stopping or slowing the attack in real time.

Another problem yields with the used software. As the sensors are deployed inside of the VM, it is possible for an attacker to tamper with the sensors, generating false data, or shutting them down completely as already described.

6. COMPARISON

On comparing those two systems, it is important to recognize that VMScope and Collapsar focus on different aspects of honeypots. On the one hand, VMScope aims to provide a tamper-resistant and invisible method to surveil honeypots while keeping a detailed view of the system. This is achieved by moving the logging component out of the host into the virtual machine monitor. On the other hand, Collapsar focuses on detecting similar attacks on multiple machines at different logical locations. It could therefore be a useful tool to evaluate the spreading of a worm over the World Wide Web or to follow the way of an attacker through a corporate network.

As the goals of both projects are disjoint and do not interfere with each other, both systems could be combined. To do this, either a Collapsar implementation for QEMU or a VMScope implementation for UML is needed. As Collapsar's changes are not as comprehensive as VMScope's, adopting Collapsar to use QEMU may be the easier task. After combining both, the Collapsar network still keeps the capability of surveilling multiple networks. But now the logging cannot be tampered with any longer, like with Sebek. This means that the combined solution is also capable of detecting attacks against single hosts. Because the virtual machine monitor needs to be modified to integrate VMScope, the source code needs to be available. Therefore the previously mentioned problem of cross-talk can be eliminated by implementing direct injection into the network interface of the virtual machine.

Although most problems of the single systems can be eliminated by combining them, one problem cannot be reduced. The performance of the combined system is most likely worse than the performance of each system on its own. Additionally to the delay introduced by Collapsar for redirecting and filtering traffic, the slowing of the virtual machine through the logging module inside of the VMM affects the combined system. As the delay between request and answer further increases, it can be even harder than for Collapsar alone to hide the fact that it is not a physical machine answering but a virtual honeypot.

As already told before, Collapsar needs a very big network to work effectively. This is also true for the combined solution. Although Collapsar is capable of detecting network scans, this should not be the main detection goal because these kinds of anomalies can be detected much more resource friendly by using the low-interaction honeypot tool honeyd. It therefore would be desirable to use Collapsar or the described combined system just for attacks, where interaction beyond the TCP/IP-stack is needed and save computation power otherwise.

doi: 10.2313/NET-2013-08-1_03

To solve this, honeyd could also be integrated into the system. For the interactivity, multiple virtual machines with different operating systems are created. Hoenyd then simulates end-hosts or even sub-networks inside the network the redirectors reside. The simulated machines should be fingerprinted as the same OSs the VMs are running. If now a simple connection initialization appears this is handled by honeyd. If further interaction is needed, the connection is then handed over to a matching virtual machine. This saves computation power because simple network or port scans are handled by honeyd. This approach can lead to problems if a fixed number of virtual machines with a specific operating system is used as all of them could be in use when another one is needed. This could be handled by dynamic creation of virtual machines from snapshots, if the data center supports this fast enough. Another challenge is the detection of still in use VMs in contrast to ones that can be reseted and be put back into the pool of available VMs.

To sum it up, both systems described above should be capable of detecting many different attacks, either on whole networks or on single hosts. Additionally the logging itself is invisible from inside the honeypots and the delay induced by the system is not detectable if the attacker is far enough away in terms of network latency. But because of legal responsibilities that operators of honeypots have to consider, it is still detectable from the outside if someone is inside of a honeypot or not. There are several methods to achieve this detection. They all include some kind of sensor—a server that is already controlled by the attacker. After intrusion into a machine, the attacker—this can be a human attacker or an automated process—sends obviously malicious traffic to remote hosts including the sensor. This can be for example another attack wave, spam mails or massive HTTP requests that look like being part of a Denial of Service attack. If the sensor receives all traffic unmodified and with the same rate the client used, the attacker can be relatively sure that he is not inside a honeypot. A honeypot operator needs to block or modify this kind of traffic because he does not know which remote machine is the sensor and if the malicious traffic is allowed to pass, the operator can be hold liable for possible damage caused by this attack[21].

7. CONCLUSION

To sum it up, both discussed systems make use of virtual machines to achieve a better logging capability than traditional honeypots. VMScope tackles the problem of circumventing or disabling conventional host-based intrusion detection systems and to provide trustable log files even if the machine itself is completely corrupted. Collapsar focuses on detecting wide-range attacks on different networks by trying to keep the single honeypot still useful to detect intrusions on single hosts.

As described in the previous section, combining both VMScope and Collapsar leads to a system which is very hard to fool as the logging does not take place inside of the virtual machines but from the outside. This system is in theory also capable of detecting distributed attacks, but this ability is dependent on how many virtual honeypots are available and where they are positioned. Currently, it is impossible to set up a honeypot that cannot be detected if the operator does not want to risk getting in conflict with applicable law. If and how this problem can be solved requires further research.

References

[1] P. M. Chen and B. D. Noble. When Virtual Is Better Than Real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, pages 133–138, May 2001.

[2] M. Dornseif, T. Holz, and C. N. Klein. NoSEBrEaK - Attacking Honeynets. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop*, pages 123–129, June 2004.

[3] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the Network and Distributed System Security Symposium*, pages 191–206, February 2003.

[4] Honeyd. `http://www.honeyd.org/`.

[5] Internet Storm Center. `https://isc.sans.edu/`.

[6] X. Jiang and X. Wang. "Out-of-the-box" Monitoring of VM-based High-Interaction Honeypots. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, pages 198–218, September 2007.

[7] X. Jiang and D. Xu. Collapsar: A VM-Based Architecture for Network Attack Detention Center. In *Proceedings of the 13th USENIX Security Symposium*, pages 15–28, August 2004.

[8] I. Kuwatly, M. Sraj, Z. Al Masri, and H. Artail. A Dynamic Honeypot Design for Intrusion Detection. In *Proceedings of IEEE/ACS International Conference on Pervasive Services (ICPS)*, pages 95–104, July 2004.

[9] Nmap. `http://nmap.org/`.

[10] J. Pfoh, C. Schneider, and C. Eckert. A Formal Model for Virtual Machine Introspection. In *Proceedings of the 2nd ACM workshop on Virtual Machine Security*, VMSec '09, pages 1–10, November 2009.

[11] N. Provos. Honeyd: A Virtual Honeypot Daemon (Extended Abstract). In *10th DFN-CERT Workshop*, February 2003.

[12] QEMU. `http://www.qemu.org/`.

[13] Sebek. `http://projects.honeynet.org/sebek/`.

[14] Snort. `http://www.snort.org/`.

[15] Snort-inline. `http://sourceforge.net/projects/snort-inline/`.

[16] Tcpdump. `http://www.tcpdump.org/`.

[17] User-Mode Linux. `http://user-mode-linux.sourceforge.net/`.

[18] Vmware. `https://www.vmware.com/`.

[19] Wireshark. `http://www.wireshark.org/`.

[20] Xen. `http://www.xen.org/`.

[21] C. C. Zou and R. Cunningham. Honeypot-Aware Advanced Botnet Construction and Maintenance. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN '06, June 2006.