Security Data Extraction Through Virtual Machine Introspection

Sarah Shekher
sshekher@uci.edu
Donald Bren School of Information and Computer Sciences
University of California, Irvine.

Abstract

With the rise of digital technology, communication and the growing availability of data on public media such as the Internet, the idea of computer security has become more prominent. Computer Security, otherwise known as "cybersecurity", revolves around protecting against, preventing and detecting unauthorized access on a system and transactions to and from it [1]. Intrusion Detection Systems (IDS) development has grown to accommodate the changes to our technology, and lifestyle [3]. The IDS gives users insight on the weaknesses of their systems while testing currently implemented security measures. The information provided can be used to further strengthen the system, thereby better protecting data within it [3] [4]. However, for this to be done, the IDS itself should be resistant to outside attack to ensure accurate feedback. Research within the field of computer security to improve the design of IDS has brought attention to technology like Virtual Machines, and hardware emulation of software. Researchers learn from the mechanism behind attacks on systems to aid in their future protection and prevention [6]. This is the ideology behind this research project.

However, as creating an optimal IDS is a large system, and would take a long period of time, the function which was focused on was one of the many thing an IDS analyzes – the processes running on a system. Eventually, this data would point to the most important processes running on a system, which are called on a regular basis. This would optimize detection of irregularity in the case of an attack. Thus, the main focus of the research was to observe the behavior of running processes through its interaction with the virtualized system calls to the operating system.

Introduction

Computer Security is improved by the use of Digital Forensics which is the act of studying corrupted data which was the result of a malicious attack. Thus, as technology continues to advance, this field has ever growing importance. It provides evidence for legal proceedings, determining vulnerabilities in a system, methods of attack, and all other exploitations to compromise a working system.[3]

The importance of computer security is best shown through the consequences of security breaches on the typical internet user. Take the PlayStation Network breach, it not only cost approximately $171 million dollars in losses for Sony [12], but also affected the lives of the gamers who could no longer use the network. Sensitive information, such as credit-card numbers, names, addresses, etc were accessible after the attack causing several cases of identity theft not to mention more than 70 million frozen accounts worldwide to limit damage [12] [13]. All this was caused by the use of an insecure platform design. The multi-billion dollar company Facebook is not safe from malware. In fact, a virus that was created specifically for social networking sites known as "koobface" has infected Facebook on more than one occasion. The mechanisms behind the attack were used to sharpen the defenses of the site, and influenced the way the corporation handled all types of malware [15]. Thus, it can be seen that it is not enough to create a general form of security; each system has its own specifications and thus, to enforce a secure environment, a tailored approach must be taken.

To define a "secure" system, three concepts are taken into consideration: security requirements, security policy, and finally, security mechanisms. Security Requirements define security goals of the system, that is, what parts of the system are to be protected, and in what way[1]. Security Policy defines the states and actions allowed, including in case of attack. For example, in certain situations, it is better to delete the data infected than to recover data that could be read illegally. If the security policy is unbroken, the system can be thought of as *"secure."* Security mechanisms are implementations that enforce the security policy, as to not allow violation and avoid deterring the practicality of the system. This includes technical mechanisms like security access protocols and encryptions, and operational mechanisms, for instance, punishment inappropriate behavior regarding data. Mechanisms must also be tested for quality assurance, as if they do not behave as required, it affects the systems functionality. It is a well-known fact that most mechanism will not work unfailingly, which leads to a goal of being mostly correct. Hence, to be secure: requirements should be defined, policies should be formed according to requirements and mechanisms should implement the policies [1] [3].

However, in order to test and strengthen the security of a system, one would have to have some way of attacking the system in a closed environment [4]. This would create an appropriate method which would not cause any lasting effect on the system at hand. In a different scenario, it would be beneficial to have an unobscured view of the system under attack so that one would have a clear idea of how the attacker could weaken the system [3]. This view would provide a stronger description of how to prevent and protect against similar attacks. One method to test the security of a system is using an Intrusion Detection System. The primary function of IDS is to attempt to detect and report whether the host has been compromised by any type of attack [3] [4]. This is done by observing any properties of the host which can be observed externally [3] [5] [6]. Encompassed in the list of properties are system calls, internal states, state transitions, and other input/output activity. If irregular activity is detected, the IDS investigate the irregularity and reports on it [4] [6].

The aim of this project was to efficiently observe inner occurrences and behaviors of processes through the interactions with the system by some means which could eventually be used to observe during an attack. The chosen frame of implementation of these means is to use a virtual machine, in a process called Virtual Machine Introspection [4].

Motivation

As mentioned earlier, Intrusion Detection Systems (IDS) are meant to capture behavior of a system and report the anomalies to the user [4]. The prime situation would be that while running the IDS in a test simulated or real corrupt situation, the original software and hardware system are left unharmed, while still accurately accumulating data on the infection. Hence, the best scenario would be an accurate large scale view of standard behavior so that the smallest difference would be visible. An IDS running on a system without any virtualization can only access the state level of hardware, which includes the physical memory, pages and registers involved with computation and the events such as memory accesses and interrupts [4] [6]. In order to be able to recreate the events which occurred during a test attack, knowledge of the operating system is required to interpret the events which occurred. This is a workable solution; however, it may end in the creation of restricted system that would be specific to the system it was made to run on. Thus, the goal of discovering generic yet strong IDS becomes a prominent endeavor.

Unfortunately, every improvement of Intrusion Detection Systems is met with increasingly sophisticated tactics to evade and defeat them [3]. Preventing such badly intended behavior is done using two means; Visibility and Isolation are key ideas for IDS development. In terms of visibility, the more the IDS can view, the more detailed the definition of "normal behavior." Basically, visibility is defined as the degree to which a system can be monitored in terms of internal state, state transitions [otherwise known as *events*], I/O activity, and other observable properties[3] [4]. The amended definition improves chances to see otherwise overlooked signs of attack as the range of analyzable activity has been increased. What this means to say is that malicious attackers would find it more difficult to attack as they cannot emulate completely normal behavior. Isolation, the second requirement, can be defined as the separation of the IDS from its host system (which could be a virtual machine). The necessity of isolation lies in preventing the IDS itself from attack by the malware being analyzed. Increased isolation causes higher resistance to an attack. Isolation is necessary to avoid the data acquired from being altered by the attack. Accuracy of data is important for research which will improve the security of future endeavors. [3][4] [6]

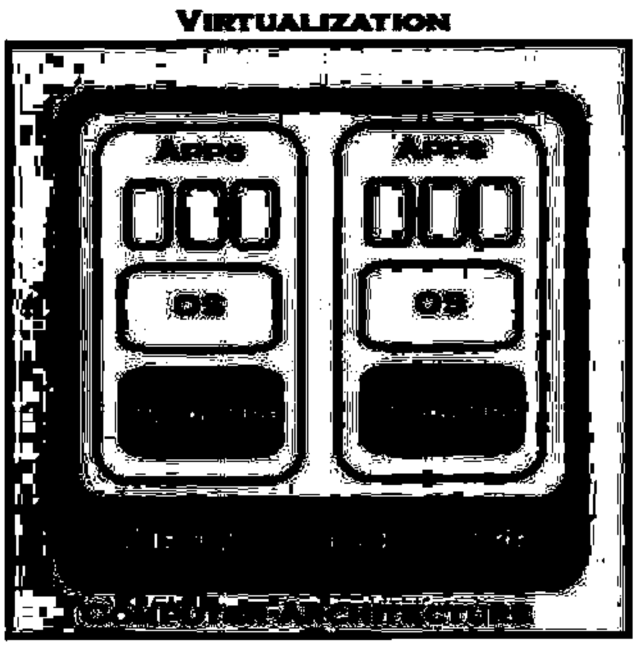
Use of IDS is not without disadvantages. The only data available to the researchers and practitioners of this field is the state of the system before the attack and the state of the system after [6] . Those in this field will essentially hypothesize the method of attack, based on previous known data on attackers, speculation and the data from the attack at hand.

Unfortunately, to truly understand the malicious attack, generally through hacking or some sort of malware, it would be best to see the process as it happens. They are left to make their analysis with inadequate information and tools that are prone to compromise, themselves. The compromise could simply be the method of stopping the attack, which is common in incident response procedure [quiescent analysis] [3] . If the events to take the system offline affect the data produced, it is no longer of practical use because of imprecision. Regular shut downs overwrite data which could be forensically important, and cutting off the power supply could cause data to be lost from the cache, or damage to the system[4]. An IDS crash would cause the system to fail open, and could either create the opportunity to compromise the system (including kernels) or require the IDS to be restarted while the application is suspended[4]. If the IDS cannot restart, then the data collected is considered erroneous.

In a similar fashion, intrusion detection systems would be ineffective if attacked or evaded by malicious software. If the IDS were meant to reside on the host, it has a clear view of events in the host software, known as visibility[4], at the price of vulnerability to attack. Attacking the IDS suggests tampering with the inner software, and would render it as useless to the testing. On the other hand, rather than attacking the IDS, the possibility malware using a much more passive approach called *evasion* [4] is greater. The malware would disguise activity so it would go undetected and still run amuck. On the host level, IDS known as Host-Based Intrusion Detection Systems (HIDS), offers a wide view of the system given that the integrity of the system is intact. An HIDS is at the risk of attack on the detection system level because of lack of isolation[4]. The other option of leaving the IDS outside the host, on the network level generally called Network-Based Intrusion Detection Systems(NIDS) provides better isolation at the cost of restricted visibility. In other words, NIDS would increase resistance to outside attack, but the inner view of the host processes would be unclear and frequent instances of missing information area commonality [4]. Thus, creators of malware see evasion as an ideal viable approach. As seen from the above evaluation, the tradeoff for visibility to the host system is the security of the IDS. In terms of intrusion detection, the peak option would be to maximize both the attack resistance and the visibility of the host. One method to maximize both the attributes, explored by Tal Garfinkle and Mendel Rosenblum, regards virtualization of the system through an approach called Virtual Machine Introspection [4].

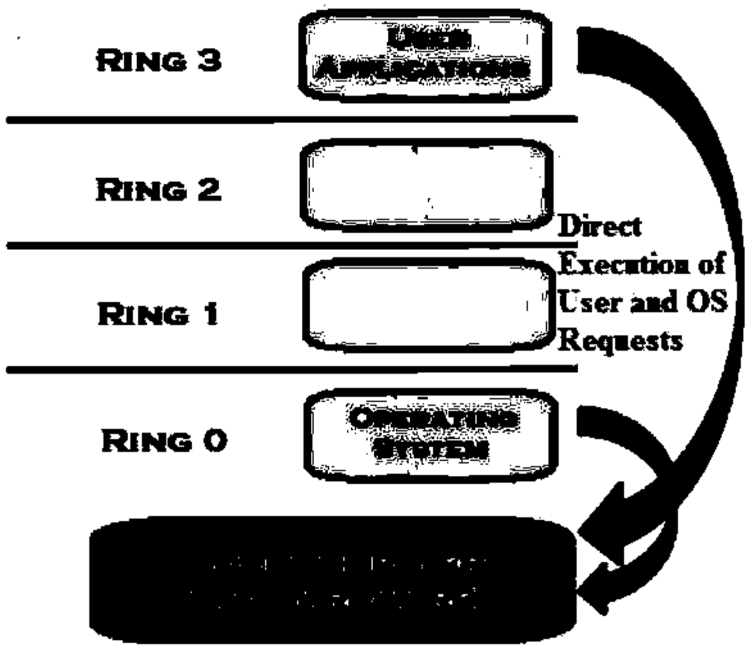
Virtualization

A field of many uses, the concept of virtualization has been in place since the 1960s. The value of virtualization has been recognized in various fields from programming languages to computer system design and has been used for purposes such as server consolidation, support for multiple operating systems, optimizations of specialized architectures [5] and, of course, security. However the true significance lies in abstraction, where virtualization proves to be advantageous [2]. Abstraction itself is important because of the sheer complexity of computer systems. Hierarchies of well-defined interfaces separate different levels



of abstraction to hide unnecessary details from applications which implement them – creating opportunities for growth in new directions. Unfortunately, the disadvantage is the limitation of such interfaces; each is designed for a specific system and specific Instruction Set Architecture (ISA) and thus may not be compatible with another interface[2]. Virtualization provides a solution to this constraint by mapping the interface and resources of the virtualized entity onto the interface and viable resources of the underlying real system, regardless of differences. Accordingly, this "real system" could be one or many virtual systems as well. [2] [8]

Virtualization, unlike abstraction, does not aim to conceal details. However, it does implement abstraction as an intermediary step during mapping from the virtual to real systems. The term Virtualization itself implies the separation of resource or request from its physical origins and delivery. In the case of operating systems, virtualization is used to create



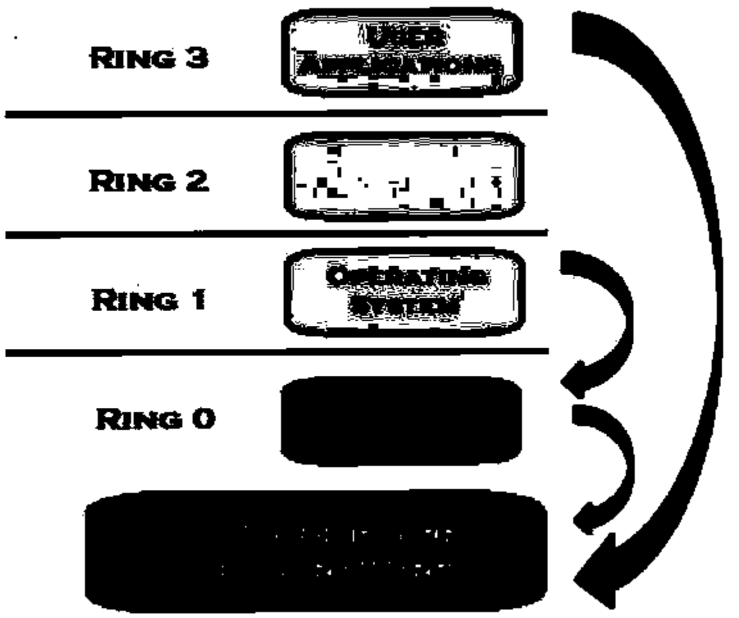
A Computer Without Virtualization

a level of abstraction for the processes which need memory resources. Virtualization gives the impression of unlimited memory, even though the underlying physical memory is limited. The allocation is done in a lower level with data swapping and disk storage [2] [8]. Essentially, virtualization is used to create a level of abstraction between the applications and the resource it is virtualizing. Benefits include unaltered user experience and improvement in management of pooled resources.

Virtualization comes in several types, as it can be implemented in several levels, such as operation system, CPU, memory etc. However, there are three main generally accepted divisions: Full Virtualization, Paravirtualization, and Hardware-Assisted Virtualization[8]. Typically, the operating system beneath the level of virtualization is referred to the host operating system, and each of the new operating systems running through means of virtualization are called guest operating systems.

I. *Full Virtualization*

A system is considered "fully virtualized" if enough hardware is simulated such that a guest operating system can run unmodified and isolation [8]. The same hardware, including CPU is utilized in this case. Through the use of a virtualization layer, the guest operating system is completely abstracted or decoupled from the hardware layer beneath it [9]. The new guest operating system is unaware of being virtualized as there are no changes to the original

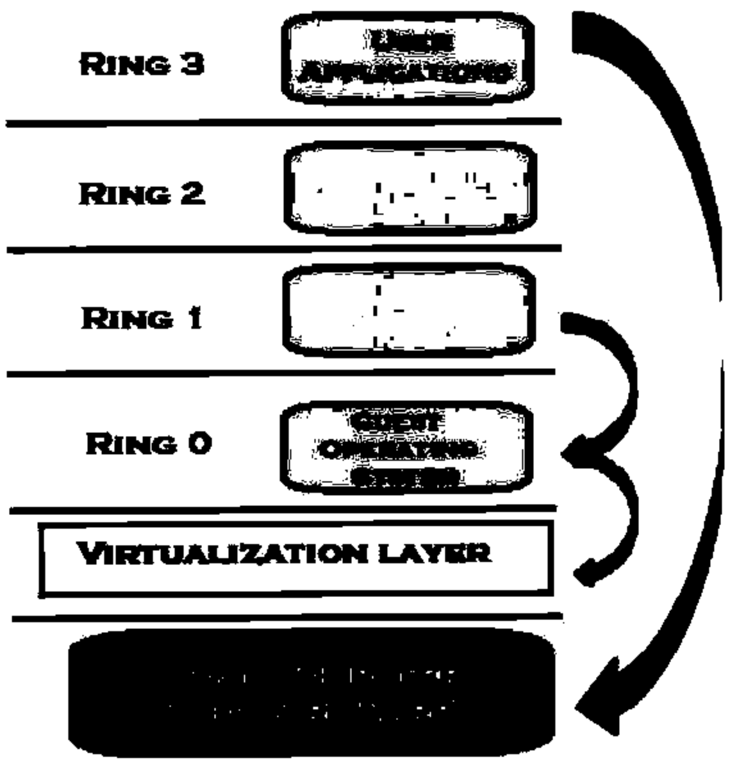


Computer with Full

Virtualization
User Apps are Directly Executed
OS Instructions are Virtualized

software; it is left as it is [2][8]. Full virtualization is the only type of virtualization that requires no assistance from the host system to virtualize sensitive and privileged instructions. It is all done through the Virtual Machine Monitor/Hypervisor.

The hypervisor, which can be thought of platform for virtual operations and supervises the guest operating systems, is a main software component of virtualization techniques. If full virtualization is implemented using binary translation, the hypervisor translates all operating system instructions dynamically, and stores the results for future uses while all user level instructions can run unaltered at their native speeds. As full virtualization is the most isolated form, it is also the most secure as well as portable [8].

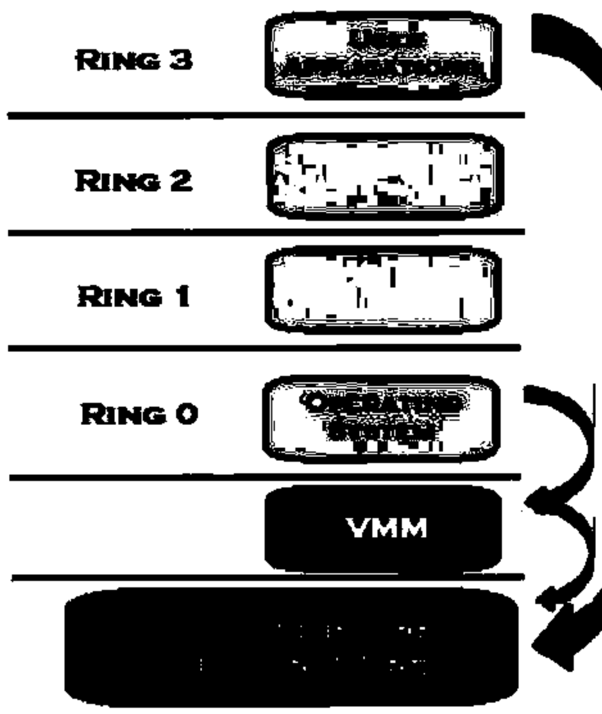


Paravirtualization
User requests - Direct execution
"Hypercalls" to virtualization layer replace the OS instructions which could not be virtualized.

II. Paravirtualization

Paravirtualization is not to be confused with "Partial Virtualization", which was the stepping stone in the process to full virtualization. Partial Virtualization simply incompletely simulates hardware, so that certain applications can be run unmodified [8]. On the other hand, partial virtualization generally cannot support an entire operating system, or all applications without any alterations. Paravirtualization is also known as Operating System Assisted Virtualization [8] [9].

Paravirtualization occurs when the hypervisor and the host operating system communicate for the sake of performance and efficiency. The host OS kernel is modified for the purpose of virtualization, and all instructions, as well as memory management, time keeping and interrupt handling operations that are unable to be virtualized are enforced through hypercalls to the virtualization layer hypervisor [8]. In this case, hardware is not simulated and a specialized Application Programming Interface that the guest OS must use.



Computer With Hardware Assisted Virtualization

Ring 0-Ring 3 are not Root Privileged Levels whereas the Virtual Machine Monitor/Hypervisor is part of Root Priviledged Level
The User Applications have access to direct execution
The OS Instructions are Trapped, and executed without use of paravirtualization or translation.

III. Hardware-Assisted Virtualization

Hardware-Assisted Virtualization is not equivalent to Hardware Emulation. Hardware-Assisted Virtualization sends guest OS calls straight to the hypervisor and any user application requests are sent directly to the physical hardware of the host OS rather than proceeding through the virtualization layer [8] [9]. This reduces changes to the guest OS as well as translation overhead seen in paravirtualization. However, it involves a great deal of CPU cooperation/support and overhead. Because of these factors, Hardware-Assisted Virtualization cannot be implemented in all architectures.

Hardware emulation, alternatively, does not require the computer to be virtualization compatible. It simply simulates/imitates the hardware wholly, and thus the guest OS can remain unmodified. In the case of hardware emulation, it is a piece of hardware of the host which is simulating a different piece of hardware while in the case of virtualization, the hypervisor which is a piece of software, imitates a particular part of hardware, or the entire computer [9] [7].

Virtual Machines

To understand virtual machines (VMs), one must understand the term machine in terms of process and system. In terms of processes, the machine does not include input/output system calls, instead the machines contains logical memory address space of the process, and the user level instructions and registers that would allow the process to be executed through its code [2]. Thus, a process VM is a virtual platform for individual process execution, and terminates at the end of the process. The virtualization software for a process VM is, in turn, called "runtime software." For systems, the virtual machine run above underlying software, and supports multiple simultaneous processes by sharing the file system as well as various other input/output resources [2] [6]. The Instruction Set Architecture(ISA) [2] [4] is what interfaces between the underlying system and the virtual machine, and all processes have real system memory and other resources allocated to them. A system VM can support an entire operating system with all its processes. The virtualizing software here is called a virtual machines monitor(VMM) or hypervisor, which was mentioned above [2] [8]. The VMM allows the guest to use a separate ISA, while the VMM emulates the hardware ISA. It provides virtualized resources to the guest system so that another operating system can function properly.

The high level reasoning for using a Virtual Machine for testing and disaster recovery is the opportunity provided to run malware within a contained, controlled and protected test environment. This holds that whatever happens within the test environment would not affect or leak into the rest of the system unless the testing processes malfunctioned [3] [6]. Since malware are of different threat levels, we can safely run it in this closed environment without any permanent damage to the underlying host software and hardware system [3]. Thus, the case of having to wipe the infected hard drive clean and reinstall a host operating system is eliminated entirely. Instead, simply create a new instance of a virtual machine and you have a clean install to replace the damaged software. Since virtual machines are software based, they can also be emulated on hardware alone [7]. This means that you can visually see what happens in the system in real time, as well control what components to monitor [5]. This will also cut back on memory costs, as the entire virtual machine would not have to be implemented, just parts significant to the needs of the user.

Virtual Machine Introspection

Using a virtual machine, knowledge of the structures which interpret the OS level semantics is available, and thus higher analysis can be achieved. Inspecting and understanding a virtual machine from outside its activities for the purpose of investigation of the software running within it is called Virtual Machine Introspection [3] [4].

Virtual Machine Introspection(VMI) is seen as a solution to the visibility and isolation trade-off. VMI IDS can directly observe hardware states and events and can deduce the software state using extrapolation. The visibility seen from this approach is comparable to the visibility of the Host-based IDS [4]. Furthermore, the data incurred from using a VMI-IDS has a smaller chance of being compromised, although visibility is maintained. VMI IDS are also strongly isolated from their hosts, which allows for detection and reporting in the face of a corrupted host. This situation proves to be more robust than "stealth" which is nothing more than hiding the IDS [4]. Isolation, Inspection and Interposition are all essential features of Virtual Machine Introspection architecture [3].

Process

"*A Virtual Machine Introspection Based Architecture for Intrusion Detection*" paved the way for VMI-related work. They discussed the situation of current intrusion detection, problems that IDS have, and how VMIs could benefit the security world. The first step was to find a virtual machine to use in order to fulfill the goal which was to observe behavior of processes running in a virtual machine with intention of discovering the normative system calls which an improved VMI IDS function would be to observe . In their paper, Garfinkle and Rosenblum created a prototype of a VMI – which is part of the high-level goal in this project as well– and they used a modified version of VMWare for that purpose. However, VMWare is not open source, nor is it free which affected its viability for this project.

Virtual Box© is a virtualization product that performs with full/native virtualization. It also has the attractive feature of "snapshots" which are instances of the guest operating system which one could revert to in the case of any type of failure [16]. It has a Graphical User Interface(GUI) which displays the image of the guest operating system and provides usability for non-technical users. The GUI also has references to all snapshots for the system. All operating systems used as guests are unmodified apart from converting the image to a .vdi file (a type of Virtual Disk Image file) [16].

The idea of snapshots is interesting, but its worth is determined by how it works. During an attack, if snapshots could be taken and later investigated, the strategy that the attacker took to attack the system would be much clearer. However, this method of using snapshots is not feasible in Virtual Box [16]. If the snapshot is malformed, then it would not run. Furthermore, the attacker could prevent the user from taking a snapshot, and the data would not be captured in real time. When a snapshot is taken, certain aspects such as the states of the virtual disks which are attached to the virtual machine in question are stored. Restoring to a Snapshot means reverting bit by bit to a previous version. Unfortunately, when a snapshot is taken, only the changes made from the last snapshot/instance are made note of. This is efficient for its purpose, but not useful in terms of getting process data at a given point of time. On the other hand, the memory at that time period is stored as well. Regrettably, not only would that be a large amount of data without direction, but it would be worthless without any way of understanding what process was using what chunk of memory. Thus, a list of currently running processes would be needed, as well as some sort of map to connect processes to their resources.

After a fair bit of research, VBoxManage was stumbled upon; it is the command-line interface of Virtual Box. It gives the user more access to inner processes of Virtual Box as well as more control over modifications to a virtual machine. While you can execute available processes through VBoxManage and set their time out periods etc., it does not have a generated list of running processes easily available. One method to retrieve such would be to use the command for the guest operating system [16]. For example, if you were running Windows XP on your Virtual Box, then the command "tasklist" would provide a list of currently running tasks/processes along with information such as their IDs, services, location etc. Microsoft also provides process status API, which could be used to discover details about underlying processes. Another method would be to tamper with the Source Code of Virtual Box. This would be messy, and cause complications. Since these methods are not generic, (as it is either based off the guest OS and that is individual to the user or changing inner components of the system) Virtual Box was not seen as successful path to the goal of observing processes.

The sub-problem became to find a virtualization system which is compatible with work towards the goal of observing processes. The next step was to learn what others in this field have been doing, and what methods they have used to implement VMIs. *"Antfarm: Tracking Processes in a Virtual Machine Environment"* is a paper which relates closely to the aim of observation of processes. This paper describes the system Antfarm, which used two different architectures [Intel x86, a fairly common architecture, and SPARC, an older RISC based architecture] to track processes in two different virtualization environments – a VMM and an emulator [5]. The Virtual Machine Monitor was Xen, an open source monitor for Intelx86. Xen has also been mentioned in **"Secure and Flexible Monitoring of Virtual**

Machines" where authors Payne, Carbone and Lee present XenAccess, a monitoring library built upon the Xen virtualization system. It is because of the analysis presented in these papers and other academic sources that Xen became the second virtualization technique used to achieve our goal.

Xen is a virtual machine monitor/hypervisor which is capable of executing multiple virtual machines with unique guest operating systems on a single host operating system. It aims to simulate close to native performance on each of these guest operating systems when in use [11]. Generally, it is used for servers. But it can be implemented on desktop systems. Xen can also be called as "baremetal" which means it becomes the first program to run after the BIOS of the host, it will then start as in "Domain-0" [6] [5]. Domain-0, or "Dom0" is a special privileged guest system which runs on the host OS and has access to resources like physical hardware, and it is mainly utilized to run the Xen toolstack for management and manage the hypervisor. The Dom0 also provides virtual disks and networks for the guest operating systems which are run as "unprivileged guests" (DomU) [5] [11].

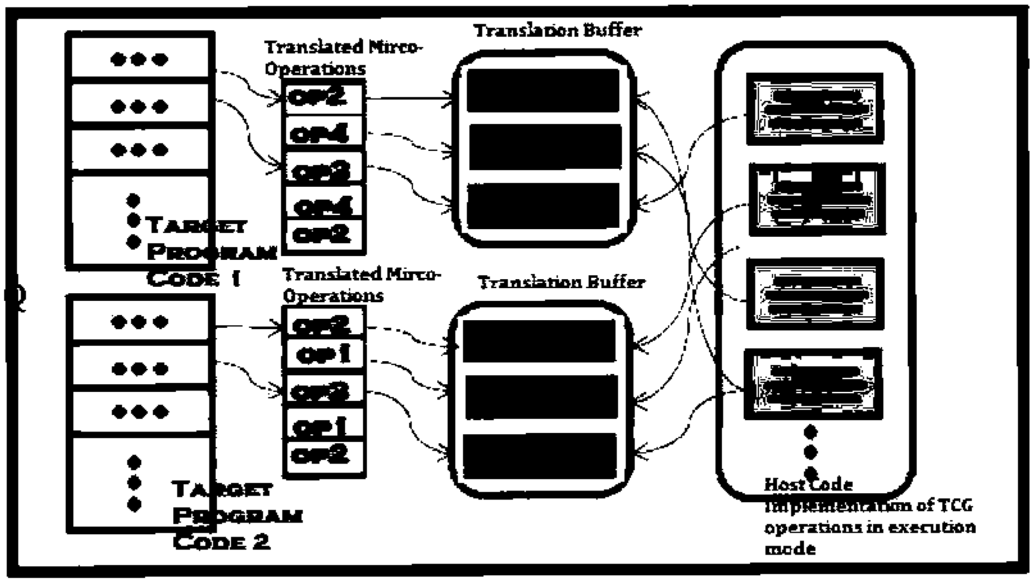
Xen offers two forms of virtualization: full virtualization, and paravirtualization [11]. Xen traditionally uses paravirtualization, and has good performance as it does not have to implement performance as well as allowing virtualization on non-virtualizable architectures (like Intel x86) [5]. Using the split domain architecture (dom0 and domU) and three forms of memory –physical, machine, and virtual- Xen can perform almost as well as a native system [6]. The drawback of using Xen is that it requires that the guest operating systems be modified and the host itself be modified with a special Dom0 enabled kernel in order to function alongside the paravirtualized version [5] [11]. In terms of full virtualization, which was done through CPU hardware emulation in this case, no modifications were required to the guest operating systems, which meant distributions of Windows were capable of running on top of Xen [11]. Consequently, because of the hardware emulation, the fully virtualized version of Xen is significantly slower. But in any case, Xen entails that the Dom0 be in some form of Linux, a change in operating systems occurred [11].

It makes perfect sense that Xen works on top of Linux distributions, as they were both created as open source software – made through a collaboration of developers from all around the world with all the source code available online. Linux can easily be modified for the user's discrete purposes. However, there are some distributions of Linux such as RedHat which have been commercialized, and are no longer cost-free. Xen provides installation guides, tutorials and resources for their Hypervisor, one of which was for Ubuntu. Ubuntu is known for its user friendly interface and stable functionality. Apart from this, Xen has been implemented on Ubuntu previously. At that period, Ubuntu was on the release 10.04 and Xen version 4.0.1 was to be used with it. It should be mentioned that until version 8.04, Xen was included in the installation of Ubuntu and was supported by developers. Presently, Ubuntu does not officially support Xen which affects the simplicity of the installation. In order to install Xen, the kernel had to be altered, and recompiled. Ideally, the rpm file of Xen would be used. RPMs are made for Linux systems, and are precompiled executables much like Window's ".exe" files. Sadly, the RPM for Xen was unavailable, and so, Xen had to be compiled from source.

Installation for Xen turned out to be much more complicated than previously conceived. Xen was incompatible with the 64-bit version of Ubuntu 10.04, so a complete and clean install of the 32-bit operating system had to be done. After the new installation of Ubuntu, Xen built, but tended to crash midway through start up. Alternatively, although Xen advertises that it can work with x86 architecture systems, the fact that the machine used did not support virtualization caused issues. Further research with the aforementioned papers showed that not only was Xen modified for their purposes, but both Antfarm and XenAccess used now outdated versions of Xen(vs2.0.6). Xen Architecture had been revised and the former xenlinux kernel was no longer used, instead, the first use of "pvops" had been implemented [11]. Especially true in open source systems, the first use of an idea is generally riddled with issues which are solved as found. Many of the issues that were faced were known and unsolved. There was a temporary solution: another popular Linux distribution known as Fedora had solved several issues found in Ubuntu. So, the obvious solution was to switch to Fedora. Unfortunately, while many Fedora users had resolved issues, they had done so in a roundabout method. Solutions were difficult to recreate particularly if the user trying to recreate was not as experienced with the inner workings of Linux systems. Fedora too had stopped supporting Xen officially in a previous release. Since many of the modifications took place in source files of the operating system excluding the kernel, implementing Xen was no longer a realistic route. Additionally, many other Linux-based operating systems had stopped supporting Xen in previous release, mainly because of the inconsistencies and bugs within the system. As Xen was not as stable as formerly anticipated, a new route was taken.

In more recent times, Virtualization stopped being supported by most commercial devices including laptops with processors by both Intel and AMD. Although there are claimed solutions to this hardware deficiency, they are generally convoluted and deal with inner workings of the system, causing "universally compatible" not to be an alternative. However, emulation is close enough to virtualization, that it can be used in similar fashions. In this light, Xen was exchanged with QEMU, a processor emulator which can be seen as a virtual execution environment generator [7] [18]. Like Xen, it is also open source and has two options for virtual behavior. The first is machine emulation. This is defined as running Operating systems and programs meant for one machine ISA on another machine. It uses dynamic translation to achieve this efficiently [7] [17] [18]. Dynamic Translation is the process of translating a line of code only when it is needed for execution [7] [18]. The second method of using QEMU is as a virtualizer [17]. QEMU reaches a level or near native performance by executing the guest code directly on the host CPU [17]. It is interesting to note that in both of the previously operating systems of virtualization, VirtualBox and Xen, QEMU is actually used to support the virtualization process. QEMU can also virtualize without either of those applications by using KVM kernel module in Linux to virtualize guests of architectures like x86, PowerPC etc [17].

Figure 1: QEMU CPU Simulation


Translated Mirco-Operations
op2
op4
op3
op4
op2
Translation Buffer
Target Program Code 1
Translated Mirco-Operations
op2
op1
op3
op1
op2
Translation Buffer
Target Program Code 2
Host Code Implementation of TCG operations in execution mode

Q

QEMU translates the guest or target machine code into the host machine code for execution [7] [18]. It is performed in a two-step procedure. First, target machine code is dynamically translated into intermediate code called micro-operations. The translation is done by the TCG(Tiny Code Generator) [7] which was originally developed as a C compiler for generic backend use. Since mirco-operations are all in C, they are compiled beforehand and executed by the host machine (most cases, x86) [7]. While work is being done to make a generic translator which would create common micro-operations, QEMU currently has predefined the mirco-operations for each emulated processor. In a virtual environment, the host machine codes corresponding to the micro-operations are stored in a translation buffer. Translation continues until a branching operation is found, at which point the host code is executed and cached in case of reuse [7].

A guide provided by Professor Heng Yin to manipulate QEMU [10] was modified for the purpose of tracing(observing the behavior of) processes running. The guide pointed in the direction of the file for translating i386 target instructions. In this simulation, the guest, that is, the target was NetBSD(another unix-based OS) and the host was Fedora 14. The first step was to compile and run QEMU, and then execute full system emulation. An important source file which handles dynamic translation in machine emulation is "translate.c". In each supported processor, a different translate.c found. Hence, in this case, translate.c within the target-i386 folder is the one to modify. Within translate.c, there is a function which is used to disassemble the instruction for translation and then puts the code into the translated code cache. This function is called repeatedly while executing a process, which would have been broken down and translated line by line. The function is known as "disas_insn" which takes in a parameter of the start of the program counter. It would not be advisable to just insert a function into the translate.c file, as disas_insn is only called at translation time. So, you predefine a trace function in a separate area(here "op_helper.c" and .h) and then have it be called within disas_insn [10]. The guide provided had premade images, and precompiled versions of QEMU, which were used initially to test if this method would be successful. After a successful attempt, the next phase involved using direct resources rather than precompiled possibly manipulated versions. Downloads from the QEMU site we used and compiled before execution. The tracer function in op_helper initially only printed an arbitrary statement every time the disassemble instruction function was called. Future stages would have included printing the program counter value, then making a tracer function which would retrieve the process ID and print it. The present research ended at researching different operation codes which an instruction was translated to.

Conclusion

Out of the three virtualization applications used, QEMU was found to be the best in terms of system compatible use, and usability. It is the most generic of the three as it does not require any modifications to the host operating system apart from installation, and can still be used for the goal of tracing and observing behavior of processes. A tracer function was implemented by another student who was added to this project. The present research provides evidence that QEMU could be used to fulfill the goal of observing and tracing the behavior of processes in a virtual environment, and future implementations of Virtual Machine Introspection Intrusion Detection Systems.

References

1. Bishop, Matthew. "What is Computer Security?" Security & Privacy, IEEE (2003): Web. 20 Nov 2010. <ieeexplore.ieee.org/iel5/8013/26429/01176998.pdf>.
2. Smith, James E., and Ravi Nair. "The Architecture of Virtual Machines." *IEEE Computer Society*. (2005): 32-38. Print.
3. Nance, Kara, Brian Hay, and Matt Bishop. "Investigating the Implications of Virtual Machine Introspection for Digital Forensics." *IEEE Computer Society*. (2009): 1024-1029. Print.
4. Garfinkle, Tal, and Mendel Rosenblum. "A Virtual Machine Introspection Based Architecture for Intrusion Detection." *Association for Computing Machinery*. (2003): Print.
5. Jones, Stephen T., Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. "Antfarm: Tracking Processes in a Virtual Machine Environment." *Association for Computing Machinery*. (2006): Print.
6. Payne, Bryan D., Martim D. P. de A. Carbone, and Wenke Lee. "Secure and Flexible Monitoring of Virtual Machines." *IEEE Computer Society*. (2007): 385 - 397 . Print.
7. Nakamoto, Yukikazu, Tatsunori Osaki, and Issei Abe. "Proposing Universal Execution Trace Framework for Embedded Software using QEMU." *IEEE Computer Society*. (2009): Print.
8. VMware. "Understanding Full Virtualization, Paravirtualization, and Hardware Assist." *VMware - White Paper* (2007): n. pag. Web.
9. Yin, Heng. "Virtualization." Syracuse University. New York, Syracuse. 17/02/2011. Lecture.< http://lcs.syr.edu/faculty/tang/Teaching/CSE791-Spring11/PPT/CSE791Cloud-0224-Virtualization.pdf>
10. Yin, Heng. "Hacking With QEMU and KVM." Syracuse University, 2010. Web. <www.ecs.syr.edu/faculty/yin/Teaching/TC2010/Proj4.pdf>.
11. "Xen Overview." *Xen*. Xen, 14/03/2011. Web. 3 Jan 2011. <http://wiki.xensource.com/xenwiki/XenOverview>.
12. Martinez , Edecio. "PlayStation Network breach has cost Sony $171 million." *CBS News* 24 May 2011, Web. http://www.cbsnews.com/8301-504083_162-20065621-504083.html
13. Seybold, Patrick. "Update on PlayStation Network and Qriocity." *PlayStation*. Sony, 26 Apr. 2011. Web. <http://blog.us.playstation.com/2011/04/26/update-on-playstation-network-and-qriocity/>.
14. Warman, Matthew. "Firesheep: Firefox extension exposes Facebook and Twitter passwords." Newspaper. The Telegraph, 25 Oct 2010. Web, 20 Nov 2010. <http://www.telegraph.co.uk/technology/news/8085354/Firesheep-Firefox-extension-exposes-Facebook-and-Twitter-passwords.html>.
15. Richmond, Riva. "Attacker That Sharpened Facebook's Defenses." Newspaper. The New York Times, 14 November. Web. 18 Nov 2010. <http://www.nytimes.com/2010/11/15/technology/15worm.html?pagewanted=1&_r=1&ref=computer_security>.
16. "VirtualBox User Manual." *Oracle*. VirtualBox, 14/03/2011. Web. 3 Jan 2011. <http://www.virtualbox.org/manual/UserManual.html>.
17. "About QEMU" *QEMU*. QEMU, 14/03/2011. Web. 3 Jan 2011. <http://wiki.qemu.org/Main_Page>.
18. Bellard, Fabrice. "QEMU, a Fast and Portable Dynamic Translator." *IEEE Computer Society*. (2005): Print.