Finding Vulnerabilities in the Oracle VM Virtualbox Hypervisor specific to CPU Attacks and RAM Attacks

Cyrus Majd

Honors Advanced Research

1/03/17

Project Abstract:

Cyber security plays a vital role in today's society and IT infrastructure. Nearly all of what we do eventually depends upon secure technologies. Virtualization claims to be the newest and most secure way to protect data and provide a secure cloud infrastructure, as they create "completely independent computer emulations". However, time and time again, malicious hackers have found vulnerabilities within hypervisors and have successfully stolen gigabytes of intellectual property from both companies and consumers alike. By exploiting and publishing key flaws in this technology before cyber thieves can, hundreds of potential threats will be eliminated.

My test procedures include stress testing both the computational and memory resources of two identical virtual machines running on the same hypervisor. My intention was to develop a methodical approach in a controlled environment in order to quantitatively determine the veracity of a cyber attack. This approach was split into two areas of study - CPU intensive attacks and RAM intensive attacks. Eventually, my results showed that both CPU and RAM intensive attacks are efficient when testing with one VM, but fail in affecting another through the hypervisor. These results are "in line" with previous studies on virtual machines, and prove that the hypervisor that I was using (Oracle VM Virtualbox) was indeed resistant to the CPU and RAM attacks I launched, although the single VM that I used was not.

Objectives:

Our three main objectives for this lab were to:

1. Observe whether or not virtual machines will break through the hypervisor (Oracle VM Virtualbox v.5.01) and share CPU power with one another when in stress when it achieves 100% utilization.

2. Observe whether or not virtual machines will break through the hypervisor (Oracle VM Virtualbox v.5.01) and share RAM storage with one another when the it is filled up.

3. Observe whether or not virtual machines will break through the hypervisor (Oracle VM Virtualbox v.5.01) and share hard drive storage with one another when their storage fills up.

Background:

An Introduction to Security

Cyber security's role in today's world is what governs most of our lives on the internet. Nearly all of what we do eventually depends upon secure technologies. It is the developer's job to make sure said technologies are safe and secure from cyber thieves and hackers, but even they make mistakes in their code; mistakes that allow hackers and cyber thieves access important files and financial data.

Cyber security is a field of study which can range from being a Certified Ethical Hacker to being a guidance counselor teaching students how to be safe on the internet. Being one of the hottest fields in the industry, cyber security experts face new challenges almost every day, whether it would be on developing new patches to faulty software, or discovering a new way a hacker could get into a system.

Virtualization is the act of partitioning the computer hardware and dedicating said partitions to a different OS (mostly) or app, creating independent virtual computers. Virtual

machines are disposable, meaning that they can be created and destroyed in software. Virtual machines can also act as proxies for large companies, and can be used as a cheap but reliable way to simulate real networks for students to study. Virtualization is NOT to be confused with hardware partitioning, since virtualization builds entirely independent compute machines. A hypervisor is the software that creates, destroys, and modifies virtual machines.

My project focuses on something a Certified Ethical Hacker would usually do; discover flaws in a specific software or system and publish them with a proposed solution. My project is about the security aspects of virtualization, specifically focusing on Oracle's virtualization technology for consumers called Virtualbox.

Identifying and publishing flaws in IT systems is very important because it helps developers to develop patches. With every new IT update there is a change to introduce security flaws or bugs. Hence certified ethical hackers and security analysts try to identify flaws in a system faster than hackers get a chance to exploit them. According to a NIST report from 2011 [6], if a "hypervisor provides a single point of security failure for all the guest Operating Systems; a single breach of the hypervisor places all the guest OSs at high risk." This just cannot be accepted in our technology centric life.

Advancements in Cyber Security

This topic has both a long and turbulent history. Ever since the internet was invented, hackers all over the world have been trying to misuse bugs and security flaws in systems, especially within virtualized cloud infrastructure, since many large businesses today rely on cloud services.

Ethical hacker organizations, such as Red Team Security, Offensive Security, CBT Nuggets, and the US Government all focus on virtualization security. However, most of their penetration tests are conducted on hypervisors developed by VMware and other prominent, expensive organizations. This leaves Oracle VM VirtualBox as a relatively untested hypervisor. For example, the last reported Nessus exploit report on VirtualBox was in January, 2016, which is VERY outdated for the IT world.

In addition, Oracle VM Virtualbox is much more different than VMware and other hypervisors because of its infrastructure on the host computer/server. There are essentially two different types of hypervisors, and the key difference between them is the way they are installed/configured. Below is a diagram depicting a common type 1 hypervisor.

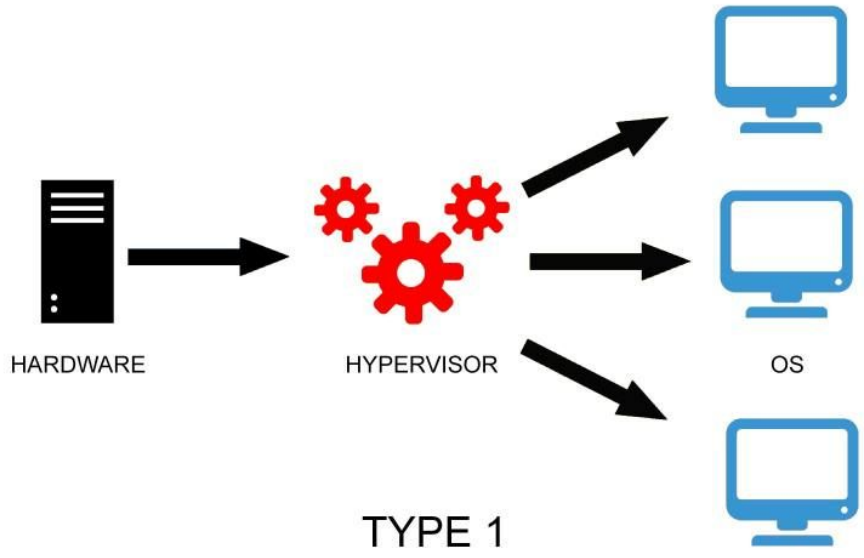


Fig.1) A visual diagram of the infrastructure of a type 1 hypervisor. As shown, the type 1 hypervisor is the next direct layer past the host's firmware/BIOS

Type 1 hypervisors are installed directly on the firmware of the host machine, occupying the next layer beyond the firmware BIOS of the machine. Type 2 hypervisors are installed on top

of a compatible operating system, such as windows. Figure 2 demonstrates this.

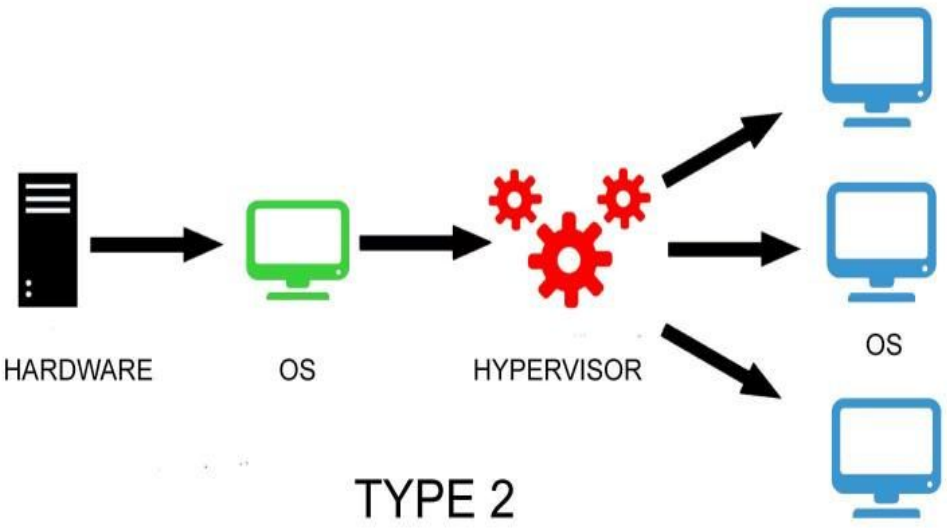


Fig 2. A diagram depicting a standard type 2 hypervisor.

Type 2 hypervisors are installed directly on top of the host OS. Oracle VM Virtualbox (the hypervisor that I will be working with) is an example of such a hypervisor.

The potential for program flaws increases dramatically with every new update. Below are a few examples of research conducted in cyber security and VMs.

Master Thesis from the University of Bergen, Norway, From R. Aarseth

A good example of such research was conducted by R. Aarseth from University of Bergen in Norway [9], who developed a comprehensive report on the security of Virtual Machines and their level of vulnerability to malwares such as Heartbleed. His results are used as reference in my research.

Aarseth began his experiments by first obtaining two different desktops, both on the same subnet. He reasons that the “timing attacks”, which according to Yinqian Zhang (a publisher of a

referenced article in Aarseth's works), makes it possible for hackers to steal encryption keys from VMs.

When he analyzed his data, Aarseth discovered that VMs may still be hacked if used on shared hardware, even if the VMs appear to be "isolated". According to his results, the methods used could potentially be "used to set up a more sophisticated attack against the users of a public cloud." He ends his report by theorizing that this may be a serious issue for almost all major public clouds.

Gartner's Report on The Security of Virtualized Servers

According to a Gartner Report from 2010 [12], 60 percent of virtualized servers will be less secure than the physical servers they replace by 2012. Gartner outlines in the report the six most common virtualization security risks and provides a recipe for how to avoid them. Gartner takes a different look at the cloud security and highlights that many of the cloud security flaws is caused because major virtualization deployment projects are being undertaken without involving the information security experts in the initial architecture and planning stages. Gardner states that virtualization is not inherently insecure. However, most virtualized workloads are being deployed insecurely, which is mainly a result of the immaturity of tools and processes and lack of good security experts.

International Journal of Computer Sciences paper on Cloud Computing Security Issues

Rabi Prasad et. al published a paper in the International Journal of Computer Science and IT Security [1], in which he outlines what cloud computing is and what are the various cloud

models and the main security risks and issues that are currently present within the cloud computing industry. This research paper analyzes the key research and challenges that are present in cloud computing and offers some of the best solutions to service providers and enterprises to avoid security failures.

The paper concludes that one of the biggest security worries with the cloud computing model is the sharing of resources. The authors discuss various models of cloud computing and security issues, and state the virtualization is major issue for cloud computing. The authors state that due to the complexity of the cloud, it will be difficult to achieve end-to-end security. They conclude that new security techniques need to be developed and older security techniques needed to be radically tweaked to be able to work with the clouds architecture.

The latest in cyber security research shows that there is a potential of threats in virtualization that may affect cyber security. Everybody agrees that the security of large cloud data centers is very important for our modern life, and with most datacenters running on virtualized platforms, it is very necessary to identify and eliminate flaws in different hypervisors.

Hypothesis:

1. A CPU attack on one of the VMs will slow down the processes of another identical VM.
2. A memory RAM attack of one VM will increase the memory utilization on another identical VM.
3. A CPU attack on one VM will fully consume all of the CPU power of that VM.
4. A RAM attack on one VM will fully consume all of the RAM power of that VM.

Experiment:

For the past year, I have been working on a security-based research project that involves discovering vulnerabilities(glitches, bugs, loopholes) in Ubuntu 16.04.1 and in Oracle VM

Virtualbox 15.01. I specifically was focusing on virtual machine “escapes”(when a VM user/file gains access to or is run by another VM without the permission of the hypervisor), and I figured that the best approach was to CPU stress test two identical virtual machines. I created two identical virtual machines, PC1 and PC2. The following are details on the two machines:

a)Ubuntu 16.04.1 Release 64 Bit

b)Base Memory/RAM-1024MB

c)Boot order: Floppy Disk, Optical Drive, Hard Disk/SSD

d)Video Memory-16MB

e)Remote desktop server DISABLED

f)Virtual hard disk encryption DISABLED

g)1 processor dedicated, 1.6 GHz clock speed, 75% execution cap

h)NAT network adapter ENABLED

i)Virtual Hard Disk boot off UBUNTU ISO IMAGE

j)Dynamically Allocated virtual storage system

The following is a list of materials used in this experiment:

A. MacBook Air (host machine)
B. Oracle VM Virtualbox 15.1 Release
C. Ubuntu 16.04.1 Release 64 Bit
D. TOP Ubuntu System Manager
E. “count” shell script
F. Memory attack shell script

I then coded a malicious script that makes the virtual machine run at 100% CPU. My plan was to stress test PC1 to see if it asks for some power that an idle PC2 has to spare. If PC2 was to donate such power-even a little- there is a vulnerability in the hypervisor, since I specifically completely isolated both virtual machines from the hypervisor.

The following is my attack code, “count”:

```
 x="$1"
echo "Process $2 started at: " $(date +%s.%N)
while [ $x -gt 0 ];
        do
                x=$(( x-1 ));
        done
echo "process $2 ended at   " $(date +%s.%N)
```

As seen above, this file first creates a variable called x. It then prints out a message when the program starts and stops in nanoseconds since POSIX Time (January 1st, 1970, 00:00:00 AM). The program then initiates a while loop that constantly repeats itself until the value of x (time in the input code below ie. 575000) is NOT greater than zero. It will then end the loop and print out when the program officially stopped.

On the first set of tests, program *count* was run at different niceness levels for in two instances with the following system inputs:

```
(bash count.sh 575000 A&);(nice -n 0 bash count.sh 575000 B&);sleep 1;ps
-l;sleep 20
```

The system input command is what runs the program (count) with specific parameters. The first part, *bash count*, addresses the program that is to be run. The number 575000 is the amount of time the program will run for (x in the "count" code). For this VM (PC1), 575000 translates to about 5-6 seconds of processing time, as intended. The time it takes to run this program differs from machine to machine, unless they both have the same **processing power** and general specs. The next part of the code, A&, simply defines a variable to this instance of the program. In this case, it is labeled as "A". The next part serves the same purpose, and nice -n "0" is the command that sets the priority of the program, or "niceness". In the above code, it is defaulted to zero. The rest of the input provides information on the CPU usage of the VM (ps -l) and also adds pause times to jot down notes (sleep 20, sleep 1). To run this code with a different niceness level, it must be typed in while logged in as an admin. This is achieved by typing in the command `sudo -i`, which lets you enter the "admin root".

Data:

For niceness = 0, both instances of count (A and B) were found to have experienced roughly the same computing time (11.81 and 11.77 respectively) after comparing the averages of five tests. The data table shown below contains the results. (fig 1)

	//Experimental Niceness: 0	//Control Niceness: 0
	B	A
start time	1477927896.802263405	1477927896.807366127
stop time	1477927908.858708537	1477927909.126208482
start time	1477928511.441055882	1477928511.451852349
stop time	1477928523.220416999	1477928522.861984691
start time	1477928668.456747769	1477928668.470043854
stop time	1477928680.211829272	1477928680.438312757

start time	1477928854.772375868	1477928854.778360000
stop time	1477928866.506867748	1477928866.647110416
start time	1477929002.956806023	1477929002.973015639
stop time	1477929014.725855770	1477929014.261576803
	Difference is stop time - start time	*Difference is stop time - start time*
Difference1	12.05644011	12.31884003
Difference2	11.77936006	11.41013002
Difference3	11.75507998	11.96826982
Difference4	11.73448992	11.8687501
Difference5	11.76905012	11.28855991
AVG of Differences	11.81888404	11.77090998

(fig 1)

The CPU distributes the tasks and divides the computing time in half. This is why the intended 6 second long programs now each take about 11-12 seconds to complete. The CPU does this because the two programs share the same niceness index and have the same code, so they are equally as demanding for resources. Next, the two processes were tested with different niceness levels. Our control group process (A) remained with a niceness of 0 (default) but our experimental group process (B) had its niceness level decreased to -5 in order to (theoretically) consume more CPU (and thus have a shorter runtime) than our control group. The results backed up this hypothesis.

//Experimental Niceness: -5	//Control Niceness: 0
B	A
1478013548.253796412	1478013548.274454900
1478013556.335801673	1478013560.078921220
1478013734.315090175	1478013734.321089798

1478013742.155983718	1478013746.101238516
1478013860.545381114	1478013860.573184259
1478013868.536137910	1478013872.110450959
1478014034.020826585	1478014034.027257412
1478014042.054679765	1478014045.947034306
1478018113.729885014	1478018113.757396859
1478018121.276924735	1478018124.916611756
Differences:	***Differences:***
8.082010031	11.80446982
7.840890169	11.78014994
7.990749836	11.53727007
8.033850193	11.91978002
7.547039986	11.15921998
Average of Differences:	***Average of Differences:***
7.898908043	11.64017797

(fig 2)

As shown, the time it took for the experimental group process to compute decreased by about 4 seconds, whereas the control group process took the same amount of time. The programs are still both running at the same time, but now one instance finishes faster than another. The CPU usage remains at around 98-100% the entire time.

The following are the results of testing with an experimental group niceness of -10.(fig 3)

//Experimental Niceness: -10	//Control Niceness: 0
B	A
1478108355.761422928	1478108355.842813791
1478108362.608876104	1478108367.880254587
1478108629.938161597	1478108630.036205762
1478108637.711882039	1478108642.684697574
1478108943.285563098	1478108943.323189231
1478108950.342262971	1478108955.276416667

1478109093.890490374	1478109093.938032693
1478109100.372786445	1478109105.655309886
1478109228.167464276	1478109228.200227203
1478109234.698023888	1478109239.706137542
Differences:	***Differences:***
6.847450018	12.03744006
7.773720026	12.64848995
7.056699991	11.95323014
6.48229003	11.7172699
6.530560017	11.50590992
Average of Differences:	***Average of Differences:***
6.938144016	11.97246799

(fig 3)

Something very interesting happens with this specific configuration. At the niceness of 0, both processes took about 12 seconds to complete, as previously mentioned. The code tested was identical for both of the processes, so both process started and stopped at the same time, but with only 50% of the CPU's computing resources for each(for a total of 100% CPU usage). At the niceness of -5, both processes were running simultaneously, but experimental group process B consumes more CPU power than when tested with a niceness of 0. The total amount of CPU usage was still 100%, as expected. However, when process B is assigned to a niceness of -10, it finished about five seconds before process A. Because it is known that process A can run at a maximum of five seconds while given 100% CPU, and because process A started at the same time as process B, it can be deducted that process B consumed ~100% CPU for 5-6 seconds to complete it's own process before handing over the full CPU power to process A. **In other words, process B seems to have prevented process A from computing anything before it**

completed itself. This is a discovery by itself, proving that it is very possible to induce a denial of service attack on this specific version of Ubuntu when logged in as a sudo root user.

As expected, the following tests conducted with experimental group niceness values of -15 and -20 respectively display very similar results.

//Experimental Niceness: -15	//Control Niceness: 0
B	A
1478893218.328216650	1478893218.321407109
1478893223.942201169	1478893229.228214130
1478893672.413309477	1478893672.405256531
1478893678.384173091	1478893683.992002030
1478893840.332820033	1478893840.331578030
1478893846.461353992	1478893851.817575111
1478892961.886865837	1478893961.878954483
1478892967.717295756	1478893972.931884809
1478894174.517993602	1478894174.511188082
1478894180.375599563	1478894185.954847456
Differences:	***Differences:***
5.61398983	10.90681005
5.970870018	11.58675003
6.128530025	11.48600006
5.830430031	11.05292988
5.857599974	11.44366002
Average of Differences:	***Average of Differences:***
5.880283976	11.29523001

(fig 4)

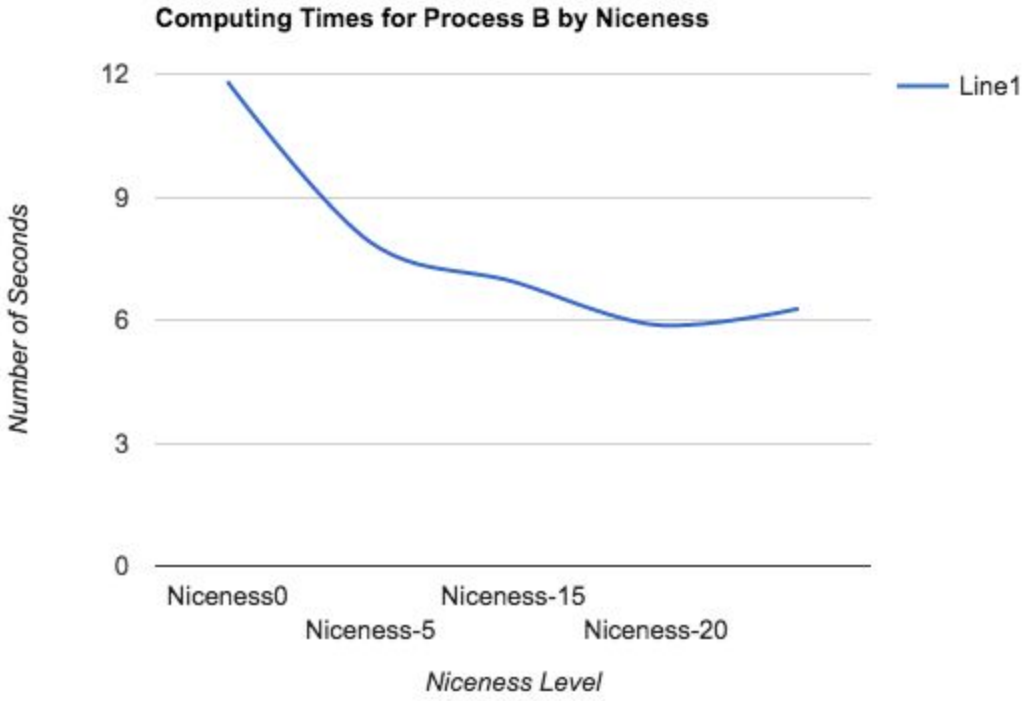
//Experimental Niceness: -20	//Control Niceness: 0
B	A
1479085662.716082866	1479085662.698009499

1479085668.198122511	1479085673.245204646
1479085748.767270061	1479085748.760568797
1479085755.279890730	1479085760.919323796
1479085807.016137317	1479085807.003721149
1479085813.357224751	1479085819.408789059
1479085856.558462184	1479085856.557135585
1479085863.019615272	1479085869.223539058
1479085896.198018431	1479085896.190087347
1479085902.768550087	1479085908.484534021
Differences:	***Differences:***
5.482040167	10.54719996
6.512619972	12.15876007
6.341089964	12.40506005
6.461149931	12.66639996
6.570539951	12.29445004
Average of Differences:	***Average of Differences:***
6.273487997	12.01437402

(fig 5)

We can see from the average of the differences in the above two tables (figs 4 & 5)that results are nearly identical to the experiment with an experimental niceness of -10. It seems as if the niceness of 10 is the cutoff line that separates the sharing of CPU resources and the hoarding of it.

The following graph demonstrates how the computing time slowly decreases and levels out. (fig 6)

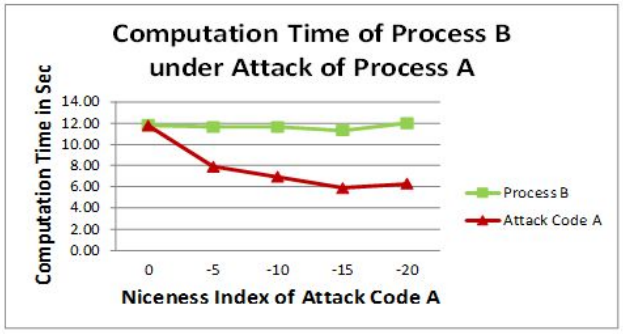


(fig 6)

This next graph shows in total the results derived from this test, along with the results from the next test. In the graph, the attack code is called process A and the user test code is called process B. (fig 7)

CPU-Intensive Attack

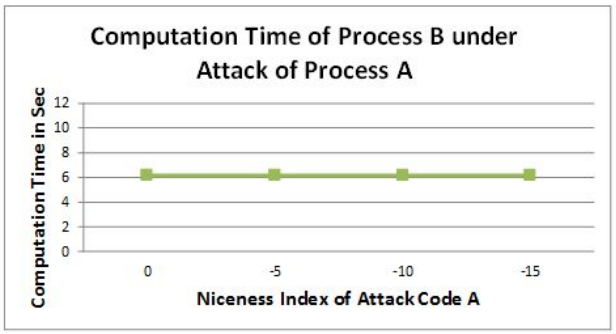
Run on the same VM



The "Control Group" for CPU-Attack Experiments:

- Attack Code A and Process B are two separate software, each one lasting for 5 seconds when run alone.
- A and B run on the same VM, sharing the CPU.
- A slows down B as the Niceness Index decreases
- At Niceness Index of -20, A freezes B for 5 seconds.
- The experiment verifies the "Operating System Leak" caused by attack code A. The attack spills over to B.

Run on different VMs



The "Experimental Group" for CPU-Attack Experiments:

- Attack Code A runs continuously on VM1 driving its CPU utilization to ~100%
- Process B runs on VM2, while its Computation Time is measured
- A does not slow down B (independent of the Niceness Index)
- A and B run on different VMs, but share the hypervisor. It means, A and B are running on the same cloud.
- No hypervisor leak is detected. The attack does not spill over to B.

(fig 7 above)

The second series of tests were conducted while monitoring two VMs at the same time (PC1 and PC2). The plan was to run count on PC2 for an infinite amount of time(forever) while then running count on PC1 at different niceness levels. Crossovers, if there were any, would be indicated by a computation time on PC1 other than 5-6 seconds.Fig. 8 shows the results for the tests with the niceness at 0.

	PC1	PC1	PC2
	//Experimental	//Utilization	//Utilization
	Niceness: 0	CPU	CPU
start time	1479089728.535449219	94%	93%
stop time	1479089734.001340316	94%	93%
start time	1479089843.050117253	93%	93%

stop time	1479089848.445491476	93%	93%
start time	1479089881.679132445	92%	93%
stop time	1479089887.000384434	92%	93%
start time	1479089919.288704391	94%	93%
stop time	1479089924.670074507	94%	93%
start time	1479089956.477907653	94%	93%
stop time	1479089961.810057752	94%	93%
	Differences:		
Difference1	5.465899944		
Difference2	5.395379782		
Difference3	5.321249962		
Difference4	5.381369829		
Difference5	5.332149982		
	Average of Differences:		
AVG of Differences	5.3792099		

(fig 8)

Unfortunately, no significant difference in computation time was found. I wasn't very lucky with the rest of the niceness tests either, as seen in figures 9, 10, 11, and 12.

PC1	PC1	PC2
//Experimental Niceness: -5	//Utilization CPU	//Utilization CPU
1479090521.794296757	96%	95%
1479090527.175383264	96%	95%
1479090569.245218864	94%	95%
1479090574.533528898	94%	95%
1479090606.078422123	95%	95%
1479090611.365465988	95%	95%
1479090637.704672527	93%	95%
1479090642.929023125	93%	95%
1479090677.938510570	93%	95%

1479090683.302848625	93%	95%
Differences:		
5.381089926		
5.288310051		
5.287039995		
5.224349976		
5.364330053		
Average of Differences:		
5.309024		

(fig 9 above)

PC1	PC1	PC2
//Experimental Niceness: -10	//Utilization CPU	//Utilization CPU
1479144699.290128550	98%	99%
1479144704.597686437	98%	99%
1479144754.766869313	94%	99%
1479144759.947665585	94%	99%
1479144790.096480499	96%	99%
1479144795.543738782	96%	99%
1479144823.554442141	92%	99%
1479144828.758479108	92%	99%
1479144859.242001610	96%	99%
1479144864.477946130	96%	99%
Differences:		
5.307560205		
5.180799961		
5.447250128		
5.204030037		
5.23593998		
Average of Differences:		
5.275116062		

(fig 10 above)

PC1	PC1	PC2
//Experimental Niceness: -15	//Utilization CPU	//Utilization CPU
1479146058.287559862	94%	99%
1479146063.646615119	94%	99%
1479146120.573801707	94%	99%
1479146126.770548664	94%	99%
1479146152.427704998	94%	99%
1479146158.131679942	94%	99%
1479146179.366264436	94%	99%
1479146185.247508683	94%	99%
1479146233.221250231	95%	99%
1479146238.549494526	95%	99%
Differences:		
5.359060049		
6.196739912		
5.703969955		
5.881239891		
5.328239918		
Average of Differences:		
5.693849945		

(fig 11 above)

PC1	PC1	PC2
//Experimental Niceness: -20	//Utilization CPU	//Utilization CPU
1479235617.728908050	94%	99%
1479235623.357723942	94%	99%
1479235690.038594346	97%	99%

1479235695.744354110	97%	99%
1479235725.523000376	92%	99%
1479235730.952676244	92%	99%
1479235763.329573481	94%	99%
1479235768.673714364	94%	99%
1479235795.152635043	96%	99%
1479235800.532483973	96%	99%
Differences:		
5.628819942		
5.705760002		
5.429670095		
5.344140053		
5.379849911		
Average of Differences:		
5.497648001		

(fig 12 above)

At this point I have tested to see if CPU intensive attacks can trigger a virtual machine escape from Oracle VM Virtualbox and if it is possible to perform a denial of service attack against a user with a CPU intensive attack with different niceness levels. Although my hypothesis that the CPU attack would create a virtual machine escape was proven incorrect, my denial of service attack undoubtedly did work as intended. These discoveries can be published as both new data (new method of denial of service attack) and supportive data for other projects (virtual machine escapes are impossible in Oracle VM Virtualbox 5.1 with CPU intensive programs). The graph on the right hand side of figure 7 depicts the results of this test. Process B is portrayed as the passive user application while process A is shown as the attack malware.

-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

The next step in my experiments was to measure any RAM overflow from either virtual machine. To do this, I first had to develop a special software that would be designed to overflow the target virtual machine's RAM buffer. I named the program "m-attack.sh", short for "memory attack bash file". The following is the source code of m-attack.sh:

```
#!/bin/bash


echo "Process m-attack started at: " $(date +%s.%N)
x="$1";
y="01234567";
while [ $x -gt 0 ];
    do
       x=$(( x-1 ));
       z=$y$y;
       y=$z;
    done
while [ TRUE ];
    do
       z=$y;
       y=$z;
    done
```

As shown above, the program's essence lies within its two vital "for loops" after receiving a user-input for variable "x" and another input for y. Variable x goes through a for loop that repeats itself as long as x is greater than zero, and with every iteration, variable y is doubled and

placed in a temporary variable z, which is then returned as the new y value after the second while loop is completed.

The charts below show the data collected while running both m-attack and count on the same machine. Note that the count process was run with the default priority level, since the test was intended to mainly analyze m-attack.

	PC1	PC1	PC1	PC1	PC1	PC1
	m-attack	count	max memory	used memory	mem-utilization	CPU util
	#of m-attack instances	time measurements of count				
start time	1	1483217249.971730208	1016252	547088	53.83%	99%
stop time	1	1483217261.196388734				
start time	1	1483221813.882945722	1016252	520156	51.18%	98.8%
stop time	1	1483221825.683517688				

start time	1	1483222023.491800332	1016252	668532	65.78%	99%
stop time	1	1483222034.581747366				
start time	1	1483222882.749087633	1016252	665512	65.49%	99%
stop time	1	1483222893.802921677				
start time	1	1483222977.769810810	1016252	618860	60.90%	98%
stop time	1	1483222989.942307685				
Difference1		11.22464991		AVG mem use:	59.44%	
Difference2		11.80057001				
Difference3		11.08993983				
Difference4		11.05384016				
Difference5		12.17249012				

(figure 13 above)

As seen above, running one instance of m-attack along with count takes up about 59.44% of total memory space, while constantly using about 99% of CPU resources. The process time for Count was around 11-12 seconds, which was expected.

	PC1	PC1	PC1	PC1	PC1	PC1
	m-attack	count	max memory	used memory	mem-utilization	CPU util
	#of m-attack instances	time measurements of count				
start time	2	1483217455.412937204	1016252	833324	82.00%	99%
stop time	2	1483217469.176163088				
start time	2	1483223148.337646924	1016252	791140	77.85%	99%
stop time	2	1483223158.758823734				
start time	2	1483223296.807678086	1016252	801140	78.83%	98.4%
stop time	2	1483223307.681619069				
start time	2	1483223406.411142142	1016252	845656	83.21%	99%

stop time	2	1483223417.508415319				
start time	2	1483223504.536051937	1016252	844488	83.10%	98%
stop time	2	1483223515.316714806				
Difference1		13.76323009		AVG mem use:	81.00%	
Difference2		10.42118001				
Difference3		10.87393999				
Difference4		11.09727001				
Difference5		10.78065991				

(fig 14 above)

In this test, two parallel instances of m-attack were launched in the background, but count was still called only once. This time, the average memory utilization increased to about 81%, which was significantly higher than the previous 59.44% utilization.

	PC1	PC1	PC1	PC1	PC1	PC1
	m-attack	count	max memory	used memory	mem-utilization	CPU util

	#of m-attack instances	time measurements of count				
start time	3	1483217841.691573674	1016252	871940	85.80%	99%
stop time	3	1483217854.569288737				
start time	3	1483218069.197858431	1016252	874228	86.02%	98.5%
stop time	3	1483218081.644736725				
start time	3	1483218200.878721236	1016252	861060	84.73%	98.7%
stop time	3	1483218213.223737597				
start time	3	1483218348.424160391	1016252	861436	84.77%	99%
stop time	3	1483218359.095939553				
start time	3	1483218468.862256321	1016252	861452	84.77%	98%
stop time	3	1483218480.975609141				

Difference1		12.87770987		AVG mem use:	85.22%	
Difference2		12.4468801				
Difference3		12.34501004				
Difference4		10.6717701				
Difference5		12.11334991				

	PC1	PC1	PC1	PC1	PC1	PC1
	m-attack	count	max memory	used memory	mem-utilization	CPU util
	#of m-attack instances	time measurements of count				
start time	4	1483218675.493513943	1016252	845240	83.17%	99%
stop time	4	1483218687.698134719				
start time	4	1483218866.356341928	1016252	844188	83.07%	99%
stop time	4	1483218877.950429285				

start time	4	1483218998.594389922	1016252	843964	83.05%	99%
stop time	4	1483219009.584689973				
start time	4	1483221365.729770340	1016252	843932	83.04%	99%
stop time	4	1483221377.997557523				
start time	4	1483221514.072874747	1016252	847872	83.43%	98%
stop time	4	1483221525.960868133				
Difference1		12.20461988		AVG mem use:	83.15%	
Difference2		11.59407997				
Difference3		10.99030018				
Difference4		12.26778007				
Difference5		11.88799				

The following charts show M-Attack's results after running on PC1 while Count was running on PC2. The number of instances of M-Attack is expected to affect the process time of Count.

	PC1	PC2	PC1	PC1	PC1
	m-attack	count	max memory	used memory	mem-utilization
start time	1	1483124777.578361805	1016252	812708	79.97%
stop time	1	1483124783.263855019			
start time	1	1483125234.141259054	1016252	765988	75.37%
stop time	1	1483125240.028196305			
start time	1	1483125312.506403961	1016252	784524	77.20%
stop time	1	1483125318.304577770			
start time	1	1483125344.375175698	1016252	777108	76.47%

stop time	1	1483125350.490849023			
start time	1	1483125514.522503396	1016252	769984	75.77%
stop time	1	1483125520.374359699			
Difference1	0	5.685489893			0.7695555827
Difference2	0	5.886940002			
Difference3	0	5.798169851			
Difference4	0	6.115669966			
Difference5	0	5.851850033			

(Above: M-Attack ran on PC1 (one instance) while Count runs on PC2)

As shown above, the first m-attack test resulted in an unusually high memory utilization for PC1. The average memory utilization is 77%, and the average computing time for Count on PC2 is about 5.867623949 seconds.

PC1	PC2	PC1	PC1	PC1
m-attack	count	max memory	used memory	mem-utilization
2	1483125689.252049214	1016252	813968	80.10%
2	1483125694.948484514			

2	1483125731.093337276	1016252	817068	80.40%
2	1483125737.023744375			
2	1483125811.091840130	1016252	817068	80.40%
2	1483125816.740025071			
2	1483125925.386134430	1016252	816340	80.33%
2	1483125931.133659217			
2	1483126103.169074872	1016252	812536	79.95%
2	1483126109.215024679			
Difference 1	5.696439981			0.8023561085
Difference 2	5.930410147			
Difference 3	5.648180008			
Difference 4	5.74751997			
Difference 5	6.045949936			

(Above: M-Attack ran on PC1 (two instances) while Count runs on PC2)

This second test seems to yield more consistent data points on memory utilization in particular. All five data points closely revolve around 80%, which is nearly a 3% memory utilization increase from the previous test. This deviation is relatively small, so there is effectively no profound change in PC2's memory utilization. Count's process time is about 5.813700008 seconds.

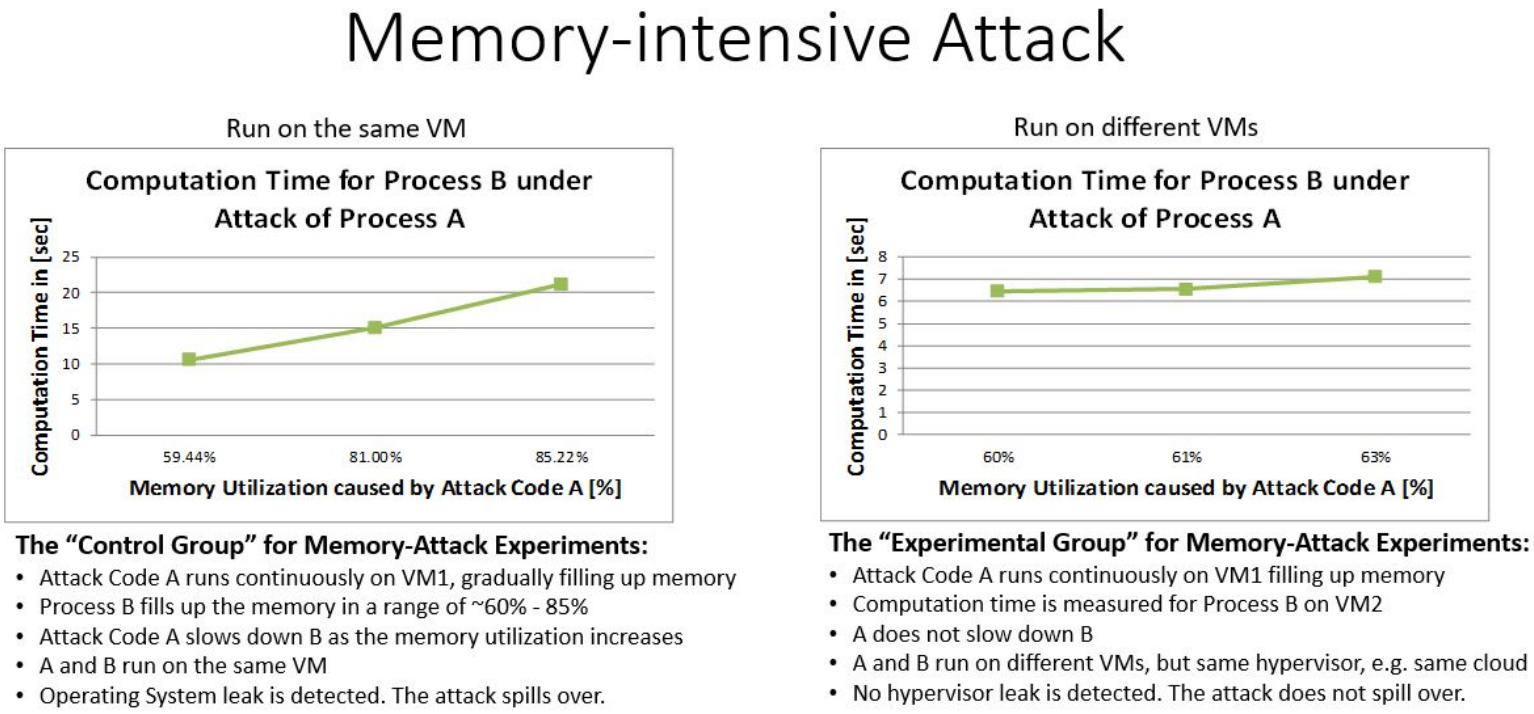
PC1	PC2	PC1	PC1	PC1
m-attack	count	max memory	used memory	mem-utilization

3	1483128566.033686735	1016252	859904	84.62%
3	1483128571.753363912			
3	1483129196.053363024	1016252	761500	74.93%
3	1483129201.992175582			
3	1483129419.229869784	1016252	762936	75.07%
3	1483129425.015265101			
3	1483129494.161291741	1016252	775148	76.28%
3	1483129499.941252810			
3	1483129551.421641325	1016252	764972	75.27%
3	1483129557.023790120			
Difference 1	5.719680071			0.7723399314
Difference 2	5.93881011			
Difference 3	5.785399914	1016272	856100	84.24%
Difference 4	5.779960155	1016272	778912	76.64%
Difference 5	5.602149963	1016272	788660	77.60%

(Above: M-Attack ran on PC1 (three instances) while Count runs on PC2)

This test above runs three instances of M-Attack on PC1 while an instance of Count runs on PC2. In this test, the average memory utilization of PC1 hits about 77.23%, and the average process time for Count is about 5.7652000426 seconds. The memory utilization never really experienced a significant change in these series of tests, which implies that there is no leak between VMs while running my memory attack code.

The results from all of the RAM Attacks are seen here, in figure 15.



(fig. 15)

Analysis and Discussion:

CPU Attacks

As observed in the test results above, the CPU attacks were indeed successful when both the test process and the malware is both launched in the same VM. As seen from figure 7, the time the benign user application took to complete remained at 12 seconds, while the time it took for the malware to complete at the highest priority level lowered to about 6 seconds. Because process A and process B were essentially the same code, they both would normally finish at ~6 seconds each when individually tested at 0 niceness(default niceness). This means that at the highest priority index, the malware code consumed 100% of the CPU's resources,

leaving no processing power to be used by the user application. This is effectively a denial of service attack, which can potentially be used to disrupt a different application's runtime. Malicious hackers and cyber thieves can potentially use this vulnerability to stifle an application running on a server in a datacenter, which will render the application (and the server) useless. This can affect many large companies that host large virtualized datacenters. A denial of service attack like this can potentially disable important programs that need to constantly be running in order for the target company to provide its services, for example.

In contrast, no vulnerabilities were found when running the CPU intensive malware on one VM while running a benign application on another. There was no change in the completion time of the user application, so the hypervisor did not allow a leak of computing power to the stressed VM. This reassuring discovery proves that Oracle's Hypervisor is secure when it comes to CPU stress-tests.

RAM Attacks

RAM attacks on one virtual machine were proven effective, since the memory utilization of the virtual machine rose from 59.44% to 85.22%(as seen in figure 15). This increase in memory utilization indicates that it is indeed possible to conduct yet another denial of service attack on one virtual machine with a memory-intensive malware. Attackers can abuse this vulnerability and effectively delay or terminate applications that are running on certain virtual machines. This can be especially damaging to VMs designed to run powerful, resource consuming simulations. An attacker can launch a memory attack like this one and stifle the processes of targeted machines by denying them access to the amount of RAM needed by those machines.

RAM attacks between two VMs, however, was proven impossible. As shown in figure 15, there was no significant change in RAM utilization in the target machine, even when the compromised VM was running the malware at the highest priority level. This shows that Oracle's hypervisor correctly manages the resources of its VMs yet again.

Conclusions:

In this research project, I investigated the robustness of the Oracle VM Virtualbox hypervisor along with individual virtual machines, and my results certainly were notable and surprising. I concluded with great certainty that even if a group of one tenant's VMs are compromised during an attempted security breach, the VMs of other tenants remain unaffected if the attack method was stressing the CPU and RAM resources of the VMs. It was surprisingly easy to compromise a single VM with my CPU and RAM attacks, but the hypervisor prevented a leak of resources between the VMs. These results closely resembled those of previous publications investigating similar attempted breaches into VMs.

Even though I did not find a vulnerability in the actual hypervisor, I certainly found a way to compromise an individual virtual machine. This find is itself a very important vulnerability that is Ubuntu-based.

Future Work:

To expand upon my research, I can investigate different types of storage attacks on virtual machines. Normally, virtualized hard drives are managed by a SAN (storage area network), which manages each VM's accessibility to that hard drive. A study on trying to find vulnerabilities regarding the SAN can be very useful for organizations with large virtualized

datacenters, since it regards the storage infrastructure of all of the virtual machines in said datacenters.

Bibliography

[1] NIST. (2011, January). Guide to Security for Full Virtualization Technologies. Retrieved October 7, 2016, from http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-125.pdf

[2] IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS) Vol. 1, No. 2, December 2011, “Cloud Computing: Security Issues and Research Challenges”, from http://ijcsits.org/papers/Vol1no22011/13vol1no2.pdf

[3] Oracle Critical Patch Update - January 2016. (2016, January). Retrieved October 07, 2016, from http://www.oracle.com/technetwork/topics/security/cpujan2016-2367955.html#AppendixOVIR

[4] Nessus Plugin ID | Tenable Network Security. (2016, January). Retrieved October 07, 2016, from https://www.tenable.com/plugins/index.php?view=single

[5] US Census Bureau Public Information Office. (2011, June 29). U.S. Census Bureau Announces Field Management Reforms to Reduce Costs and Enhance Data Quality - Miscellaneous - Newsroom - U.S. Census Bureau. Retrieved October 07, 2016, from https://www.census.gov/newsroom/releases/archives/miscellaneous/realignment.html

[6] InfoWorld. (2013, December 18). 4 ways network virtualization improves security. Retrieved October 07, 2016, from http://www.infoworld.com/article/2609571/networking/4-ways-network-virtualization-improves-security.html

[7] Exploitdb. (2014, October 8). VirtualBox - 3D Acceleration Virtual Machine Escape (Metasploit). Retrieved October 7, 2016, from https://www.exploit-db.com/exploits/34334/

[8] Falcon, F. (2014, June 27). Breaking Out of VirtualBox through 3D Acceleration. Retrieved October 07, 2016, from https://www.coresecurity.com/corelabs-research/publications/breaking-out-virtualbox-through-3d-acceleration

[9] Aarseth, R. (2015, September). Security in cloud computing and virtual environments. Retrieved October 7, 2016, from http://bora.uib.no/bitstream/handle/1956/10695/138625252.pdf;jsessionid=F32AF4096B3F0FFEA838BFF5F278879F.bora-uib_worker?sequence=1

[10] Roguine, S. (2014, January 6). 7 Predictions for Virtualization in 2014. Retrieved October 07, 2016, from http://www.acronis.com/en-us/blog/posts/7-predictions-virtualization-2014