Malware Virtualization Analysis

DYLAN FAHR

7/17/16

**Table of Contents**

**Executive Summary**

**……………………………………………………………………….…. 2**

**Purpose**

**………………………………………………………………..…….. 3 - 4**

**How to Detect a Virtual Environment ………...………………………………………………..…………......... 5**

**Experiment**

**……………………………………………………………….................. 6**

**The Program …………………………………………...…………………………. 7 - 9**

**Conclusion**

**………………………………………..………………………….. 10 – 11**

**Appendix**

**………………………………………………………………………… 12**

**Executive Summary**

This exercise serves to demonstrate the key environmental differences between a virtual and host environment. By creating a program that is able to differentiate between the two environments, one can have a better understanding of key differences between the two and how a malicious program can be coded to detect its environment. With success in this exercise, the user is able to further understand the differences between the host and virtual machine and able to understand how malware can differentiate the two, and as a result find a way to prevent malware from detecting the difference.

Operating Systems Used:

* Virtual Machines: Fedora 24 Workstation, Ubuntu 16 LTS

Virtual Machine Software:

* VMware Workstation Pro 12

Software Used:

* Sublime Text 3

Misc. Software:

* Snipping Tool (for screenshots)

Conclusion:

In conclusion, this exercise further demonstrates the difference in environmental variables of a virtual environment and a physical environment by identifying the differences between environments and key features between the two, an understanding can be made on how two are different. Additionally, by having to create a program that can the difference between the two, this environmental difference is further exemplified and utilized in order to understand and engineer malware and consequently discover ways how to prevent malware that seek for either environment and are self-aware.

**Purpose**

The purpose of this exercise is to not only further understand the differences between environmental variables of a virtual environment and a host environment, but also to understand how a malicious program can be made in a way to detect its environment and differentiate between the two, which can be utilized in malware prevention. By coding a program, malicious in intent, to detect the environment it exists in and differentiate between a virtual environment and a host environment, the user is able to observe the key differences between the two environments.

As stated before in previous experiments and reports, the difference between a host and virtual environment exists in many ways. The difference can be noted in performance, hardware makeup, driver names, and the files listed within the operating system. By having these differences, the machine is able to tell the difference between the two environments when running the hypervisor. The machine running the hypervisor is able to detect that it is both running the host operating system and the virtual environment through the use of a hypervisor. Without the usage of a hypervisor, or without the various differences between how the environment runs and is built within the hypervisor, the machine can be fooled to believe that is running a second operating system within itself, which may clash with the original host operating system.

By having the differentiation between the two environments, the goal of isolation and security can be guaranteed. By having a virtual environment containing various variables and values that are separate from an operating system operating directly on the host machine, the effects of all functions and operations upon the virtual environment are confined to its own boundaries. For example, if a program were to run within the virtual environment that effected the visual driver of the machine, then the program will be confined to the visual driver of the virtual environment rather than the host visual driver. This is due to a different value in variables for the drivers within the virtual environment. By having different variables, process, and drivers within the virtual environment, all action within the virtual environment is confined within its boundaries and only effects those particular drivers.

By coding a program that can differentiate between a virtual and host environment, the user is able to understand how a malicious program can become self-aware of its environment, and the user will be able to discover ways to prevent it. In this exercise, the differentiation between the two environments is further explained and demonstrated through the building of the program. Various variables identified and targeted in order to detect the differentiation between environments is utilized. By focusing on these variable the user understands what can be utilized in order to detect the virtual environment and also how to fool the program. By looking at how a program can differentiate between the two environments, key environmental differences become evident. One key difference Is how various drivers and hardware of the virtual environment is renamed or removed depending on the hypervisor platform. Another key difference is how processes are changed and the ids progressively changing each time the virtual machine is powered on. With these differences, not only is isolation guaranteed, but it becomes evident that there is an environmental difference between the host and virtual environments. Those differences can be referred to and used in order for the program to detect its environment. Additionally, the user can also notice these key differences and notice as well, how to change the environment and fool the program, thus differing it from detonating in the host environment, and containing it within the virtual environment.

**How to Detect a Virtual Environment:**

As stated before, virtual environments guarantee isolation, confidentiality, and security by creating unique processes and drivers that are unique to the environment. These unique drivers, processes, files, etc. all are key in creating a virtual environment that is separate from the host environment but does not appear or function as if it’s just a normal program, but rather performs as if it is a separate machine entirely.

A virtual environment is able to operate as a fully independent machine because of the environmental variables created by the hypervisor. As observed from other reports, the hypervisor is able to utilize various hardware and software from the host system in the creation and operation of the virtual machine. The virtual machine has almost the same elements as the host system with very little differentiation between the two (depending on the hypervisor used). This creates a virtual machine that operates as if it exists on a separate machine entirely and not just a program within another machine. The virtual machine is able to perform the same sort of functions as the host environment with little difference between the two. This is key in order to perform various functions within the virtual machine that may not want to be performed within the host system.

The virtual environment uses its own unique hardware and software that is different from the host system but is based upon it. The hypervisor creates various hardware and software that are based on the host machine’s but is renamed or created independently in order to ensure an isolated environment that is separate from the host system. An example of this differentiation, which is observed in prior experiments, is that the hardware and drivers of the virtual environment are given unique variables and names that help it identify as part of the hypervisor and not the direct elements of the host system. This helps guarantee the concept of isolation and security of the machine. By having all processes confined and directed to those created and listed within the virtual environment, all functions and operations within the virtual environment are confined within the hypervisor and is made more difficult to leak onto the host machine,

By seeking out a way to differentiate between the environments, one must look at the environmental variables listed within the virtual environment. A key way to determine if the environment is virtualized or not is by looking at key variables such as the processes running and the hardware and software installed in the environment. If a program were to be able to seek out hardware that is labeled differently within a virtual environment, for example a hard drive is labeled as virtual drive, then the program is able to determine it exists within a virtual environment. The same can be said for process being run or particular files that exist within the environment that don’t exist in the host.

**Experiment:**

For the experiment, the Linux-based operating system was chosen and tested with the program in order to detect if the environment was virtualized or not. Upon initial examination of the virtual environment, prior to the creation and deployment of the program, several key features were noticed within the virtual machine. As outlaid by previous experiments and reports, the hardware and drivers of the virtual environment had different labels. The labels of the hardware contained the word “virtual” or “VMware” within its title. This is a telltale sign that the environment is virtualized within the VMware Workstation hypervisor. Another attribute that was noticed was the presence of the VMtools file within the system. Lastly, another attribute observed was that the number of cores used by the CPU was different then the amount used by the host system. These attributes were factored in and utilized in the creation of the program and the detection of the virtual environment.

By having the program search for the existence of the VMtools file, the program can be able to determine that it was running within a virtualized environment within the VMware hypervisor. The VMtools file exists as a default tools configuration file which enable the core function to occur from the host machine to the virtual environment. This can include keyboard command, drivers, and various input that the host machine can have upon the virtual environment. This file is key towards the overall operation and configuration of the virtual environment within the hypervisor and is key in differentiating the environments from each other.

The detection of the number of cores of the CPU is also key towards differentiating between the environments and determining if it is virtualized or not. Upon normal creation of a virtual machine, VMware workstation hypervisor defaults in creating one processor with one core. This is so that the virtual machine does not draw more on the resources and cores of the CPU, thus hindering performance of the host and allowing it to perform other functions while virtualizing an environment. By have one core active and present within the virtual environment, the environment is able to function properly and fully in order to present itself as a separate machine from the host. With detection of this default amount of cores or a lesser amount of CPU cores available on the system, the program is able to make a more concrete decision as to what environment it exists on and how to function accordingly.

**The Program:**

The purpose of the program is to create something that is able to seek out key features of a virtual environment and is able to determine which environment it exists within and how to act accordingly. The features that were detected in the Linux environments to help the program detect which environment it exists in were the presence of the VMtools file and the number of CPU cores available within the machine.

The program, coded in C, was designed with the ability to detect particular file locations and through those file locations, to detect the presence of the key features of the virtual environment. The first task was for the program to search for the existence of the VMtools file. The VMtools file is a unique file that solely exists within a virtual environment that is created within the VMware workstation hypervisor. Various other hypervisors have other configuration and tool-base files within their virtual machines, but for the sake of simplicity and effectiveness of the experiment, the VMtools file was chosen. Upon further search within the virtual machine, it was found that the file exists within the bin file of the main folder of the virtual machine. by having the code:

char \*lFileName = "/usr/bin/VMtoolsd";

the file is assigned is the direct location of the VMtools file. This value is passed to the function fileCheck which further determines if the file exists or not and what to do if it is detected.

void fileCheck(const char \*lFileName){

if(!access(lFileName, F\_OK )){

printf("The File %s\t was Found\n",lFileName);

printf("This is a virtual machine");

vAttack(false);

}else{

printf("The File %s\t not Found\n",lFileName);

vAttack(true);

}

The program also searches for the existence of the CPU and searches for the number of the cores within the CPU. This section of the code was inspired by the code written by my group mate, Josh Willis, who developed several methods of accessing the main folder that contained information of the system, and used the information in order to determine the environment.

int numCores(){

int cores = 0;

char vInfo[50] = "/proc/cpuinfo";

FILE \*cpu = fopen(vInfo, "r");

char buff[50];

fscanf(cpu, "%s", buff);

while(strcmp(buff, "cores") != 0){

fscanf(cpu, "%s", buff);

}

fscanf(cpu, "%s", buff);

fscanf(cpu, "%d", &cores);

return cores;

}

The way how this function operates is by accessing the process folder within Linux distributions and using the information inside. The /proc/ folder contains a vast amount of information regarding the hardware and drivers running within the machine. For the sake of the experiment, the cpuinfo subfolder was chosen in order to help determine the number of cores used by the CPU in the environment.

The file within the folder is opened and is read and scanned. A buffer variable is created and utilized to help sift through the information of the file to find the information regarding the amount of cores used. The buffer is compared to the term, “cores” to help find the exact pinpoint of the information regarding the cores of the CPU and help return that value. The information about the cores in the CPU is found and scanned and returned to the main part of the program.

After information is collected from both procedures of the program, a Boolean value is passed to a function within the program which conducts the attack. For the sake of simplicity, the attack is a simple infinite loop of text that, when used maliciously, can overload the system and eat up system resources in order to continue the loop. The attack could be changed into various other forms of malicious attack but in order to preserve simplicity of the experiment, this route was chosen instead.

The attack function of the program is passed a Boolean value from both the fileCheck function and the main function of the program. The program is detailed as such:

void vAttack(bool result){

int i = 0;

if (result = true){

while(i == 0){

printf(" Never gonna give you up / Never gonna let you down / Never gonna run around and desert you");

}

}

else if (result = false){

printf("Nothing to see here");

}

}

The function is passed a Boolean value which is identified as the variable name, result. If the result was true, that the machine is a target machine, a physical or host machine, the attack commences. The simple attack is commenced by having an integer value of 0 being used in a while loop. While the integer value remains at 0, the text of the attack is displayed and repeated until the loop ends, which is indefinite. This attack uses up system resources and if used effectively, could have devastating effects if designed differently. In the situation of the value of false being sent to the function, the program displays a prompt that explains that it is not malicious in anyway, thus not commenting the attack.

This simple difference in Boolean values passed to this function, determines if the program detonates or not within the environment it currently is located within. If this attack were programmed differently in order to detonate a more malicious payload, it would deploy accordingly to the environment it was within and carry out the attacker’s desired action upon the victim computer.

**Conclusion:**

In conclusion, this exercise further demonstrates the difference in environmental variables of a virtual environment and a physical environment. By designing a program that can detect the environment it is within, the user is able to observe the differences between environments and allowed to use the program accordingly.

Virtual environments have unique environmental assets that help identify it as a virtual environment from a physical environment. By having various processes, hardware labels, and files that are unique to the virtual environment and the hypervisor that hosts it, the virtual machine is able to identify itself as an independent machine and spate processes within the physical machine itself are able to identify it as a separate program. By having this sense of isolation of the virtual environment from the host machine, the virtual machine is able to function as an independent operating system and all interactions and functions within the virtual environment remain within itself and not easily leaked to the host machine. This is key in preventing malicious attacks which could seek out to detonate in various machines and be wanting to spread to either multiple machines and / or deeper within the machine itself. By having the virtual environment have independent processes and attributes in order to function as a separate machine, the malicious attack is confined to the virtual machine and upon deletion of the virtual machine is effectively eliminated and contained in said machine.

Looking at the virtual machine, two key environmental variables stand out, the presence of the VMtools file within the environment and the number of cores used by the CPU. The VMtools file is unique to the use of virtual environments within VMware hypervisors and serve as a tell-tale sign that the environment is virtualized. By detecting this file, a program is able to instantly tell which environment it is in and can act accordingly. With the information gathered from the CPU and how many cores are being utilized, the program is able to determine whether the CPU is from a virtual environment or from a physical environment.

What can be learned from detecting environmental differences between the virtual and physical machine, the user can not only program the malicious program to operate accordingly, but to also understand and to prevent future attacks from occurring. Since the environmental features that are outlined in the excursive is that the drivers, hardware, and files of the virtual machine is unique to the virtual machine, it can be determined that a way to prevent a program of detecting the virtual environment separate of the host environment, is to rename the drivers and to change the virtual machine accordingly. Two key examples of this are attributed to the two features targeted by the exercise, the VMtools file, and the number of cores available. By reading the VMtools file, along with other files that unique to the hypervisor and the virtual environment, a simple file comparison procedure by a program can result in false results where the files are not detected. Another way around the detection of the files are to encrypt the files or to securely lock the files to where certain access permissions are limited to particular users and are further made secure. Dealing with the issue of the cores used by the CPU in the virtual machine is by changing the configuration of the virtual machine and to change the number of cores made available to the virtual machine. By having equal or greater amount of cores to that of the host machine or other commercial machines, the user can fool programs that look for lower number cores and tricks the program to believe it is in a physical machine. By tricking the program, the malicious payload is contained within the virtual environment and can be deleted in order to prevent further infection.

In the end, it can be gathered that the difference between environments are very much apparent and that programs can be made or modified in order to detect the difference between the two. By having a program, especially a malicious program, being able to differentiate between a virtual and physical environment, the program becomes self-aware and the creator of the program can detonate the program accordingly to the environment desired. This proves as a key issue in the realm of computer security and information assurance and security where virtualization is utilized in a growing trend. Fortunately, the virtual environments can be configured and changed accordingly in order to prevent attacks or the detection by malicious programs that seek either virtual or physical environments. By understanding the difference between environment and how programs that seek the difference between the two, users can have a beater knowledge and understanding of how these malicious programs operate and how to accordingly stop them and trick the malicious programs into detonations within false environments.

**Appendix:**

This code identifies the presence of the VMtools file and the number of cores of the CPU, then determines whether it is a virtual machine or host environment.