VIRTUAL MACHINE INTROSPECTION TOOL DESIGN ANALYSIS

Justin Martin
Michigan Technological University, justinma@mtu.edu

VIRTUAL MACHINE INTROSPECTION TOOL DESIGN ANALYSIS

By

Justin Martin

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2022

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Computer Science

Report Advisor:	*Jean Mayo*
Committee Member:	*Timothy Havens*
Committee Member:	*Craig Einstein*
Department Chair:	*Linda Ott*

Table of Contents

Abstract

Virtual machines are an integral part of today's computing world. Their use is widespread and applicable in many different computing fields. With virtual machines, the ability to introspect and monitor is often overlooked or left unimplemented. Introspection is used to gather information about the state of virtual machines as they operate. Without introspection, verbose log data and state information is unavailable after unexpected errors or crashes occur. With introspection, this data can be analyzed further to determine the true cause of the unexpected crash or error. Therefore, introspection plays a critical role in portraying accurate historical information regarding the operating of a virtual machine.

The Virtual Machine Introspection Tool is a tool created for monitoring the Virtual Machine Platform. This tool is the main contribution of this project. Its goal is to accurately store and present historical state data regarding the Virtual Machine Platform. The tool seeks to provide an intuitive and easy to understand user interface that interprets and displays the stored historical state data. This tool interfaces directly with the structures of the Virtual Machine Platform, which are not accessible to other virtual machine monitoring tools and introspection frameworks. This exclusive access to the Virtual Machine Platform's structures is the main motivation for the development of the Virtual Machine Introspection Tool.

This report gives an in depth analysis of the design and implementation of the Virtual Machine Introspection Tool. First, the importance of virtual machines and virtual machine introspection is introduced. Second, background information and common concepts are explained. Third, related work is discussed and compared to the Virtual Machine Introspection Tool. Fourth, the design and implementation of the tool is explored. Finally, future work and future potential additions are discussed regarding the Virtual Machine Introspection Tool.

1 Introduction

Virtual machines are an important resource used commonly in the computing industry for many different applications. A virtual machine allows its user to emulate a different computer system within the same physical computer without disrupting the accessibility or integrity of the original system. Virtual machines are designed to allow one or multiple guest machines to run on one single physical host machine with the benefits of processing scalability and isolation. This concept allows for sandboxed environments which can scale to the computing needs of the user. It is important to have the capability to introspect and understand the running operation of virtual machines. Without this capability, the virtual machine is left to be monitored by the host's operating system, which may not produce an accurate depiction of the system's runtime status.

Virtual machine introspection can be performed in a multitude of ways and has many different uses. Virtual machine introspection has been used in intrusion detection [1], digital forensics, system resource usage tracking, kernel integrity, anti-malware [2], firewall, virus scanning, cloud management, cluster patch management, and similarity clustering [3]. Introspection can perform a critical role in the analysis of virtual machine state via many different avenues. For instance, introspection tools can interface with virtual machines and expose "a raw byte-level VM memory view" [3]. Some of these same tools can then present that memory information to the user in an organized and user-friendly manner.

There are two main approaches to introspection that many of these methods follow. There is the concept of pause and introspect and the concept of live introspection. The main difference between these two approaches is that the pause and introspect method only attempts to read into virtual machine state after virtual machine operation is halted. On the other hand, live introspection occurs while the virtual machine is running, providing a real time interpretation of the state of the virtual machine.

The Virtual Machine Introspection Tool aims to take advantage of the concept of pause and introspect to interpret and display a representation of the state of a virtual machine created with the Virtual Machine Platform. By using direct access to the structures of the virtual machine when it pauses operation, the tool collects data about the virtual machine. This access cannot be achieved by outside tools due to strict access rights to the Virtual Machine Platform. Tight control over the integration of the Virtual Machine Introspection Tool and the Virtual Machine Platform needed to occur so that access to the structures of the Virtual Machine Platform remained restricted to only the Virtual Machine Introspection Tool.

The Virtual Machine Introspection Tool performs introspection by first interpreting virtual machine state, then storing into csv files and later displaying the data in the form of a table. This information can later be used to construct a history of the operation and state of the virtual machine. The Virtual Machine Introspection Tool also provides a user-friendly interface to this state data gathered during introspection.

2 Background Information

2.1 Common Concepts

Virtual machines work very closely with the operating system to gain access to hardware resources. The goal of virtualization is to present a virtual environment that mimics the behavior and operation of a real environment. To create this environment, operating systems separate virtual machine operations and structures from real ones by partitioning and aggregating hardware and software resources [4]. Multiple virtual environments can be created, maintained and managed by the operating system which is handled by a virtualization layer within the operating system's architecture. This layer is also referred to as a virtual machine monitor [4] or as a hypervisor [5].

Most modern operating systems implement memory isolation to ensure that user programs do not have direct access to hardware resources [6]. This concept can be explained as two separate spaces of operation: the kernel space and the user space. The user space is where all user driven software runs. Processes running in user space are limited in capability by the operating system. The operating system uses the concept of virtualization to provide access to physical resources for programs running in user space. The operating system limits the areas that user programs can access by providing an illusion of a physical memory space. User space processes can make requests to kernel space in order to run privileged commands, which are run on behalf of the user space process by a kernel process. Modern processors typically use a "supervisor bit" which is set when a user process needs to make a request to kernel space [6].

2.2 Virtual Machine Monitors (VMM)

The virtual machine monitor is a layer of software that can export an abstraction of the virtual machine that resembles the hardware very closely [1]. The VMM virtualizes the hardware to allow multiple virtual machines to run simultaneously within the same system. VMMs typically run in kernel space as a kernel process in order to gain direct access to the physical resources. Hosted VMMs such as VirtualBox or VMware run VMMs alongside the host operating system and they operate by wrapping code and using drivers to interface with the host operating system's VMM. When a virtual machine monitor or hypervisor creates a new virtual machine, the newly created virtual machine is referred to as a guest operating system. The original operating system that provided the capability to create this virtual machine is referred to as the host operating system.

This work focuses on taking advantage of the Kernel-based Virtual Machine which is also known as KVM. KVM operates as a subsystem of Linux and creates separate processes for each virtual machine [5]. Specifically, a virtual machine is created by opening a new device node named `/dev/kvm`. This device node is a file which communicates with user space programs via `ioctl()` system calls. User space programs are able to make requests to KVM by specifying which specific device node to communicate with in each `ioctl()` call [5] [7]. KVM provides the capability to create a new virtual machine, allocate memory to it, run instructions, read and write to vcpu registers, and to interrupt vcpu operation [5]. These features integrate closely with the Linux kernel which allows KVM to use pre-existing kernel functions and structures to perform these virtual machine operations [5].

2.3 Virtual Machine Platform

The Virtual Machine Introspection Tool performs introspection of the Virtual Machine Platform which is similar to the open-source machine emulator QEMU [8]. QEMU allows the user to create and run a new operating system on their host machine. This new operating system is contained within user space and can execute applications that a host machine running this operating system can execute [8]. QEMU works directly with KVM or Xen to perform address and code translation from the guest operating system to the host. The Virtual Machine Platform makes use of KVM to provide this efficient access to virtual resources. This platform is an x86-based platform that takes assembly instructions as input. The platform leverages KVM to handle all privileged virtual machine operations.

The Virtual Machine Platform provides methods to create a virtual machine, configure it, and to run instructions on this virtual machine. This platform maintains virtual structures which simulate the operations of physical hardware. KVM is utilized to make requests to the kernel to run instructions or change aspects about the virtual machine on behalf of the platform. Virtual machines created by this platform only support running programs written in x86 assembly.

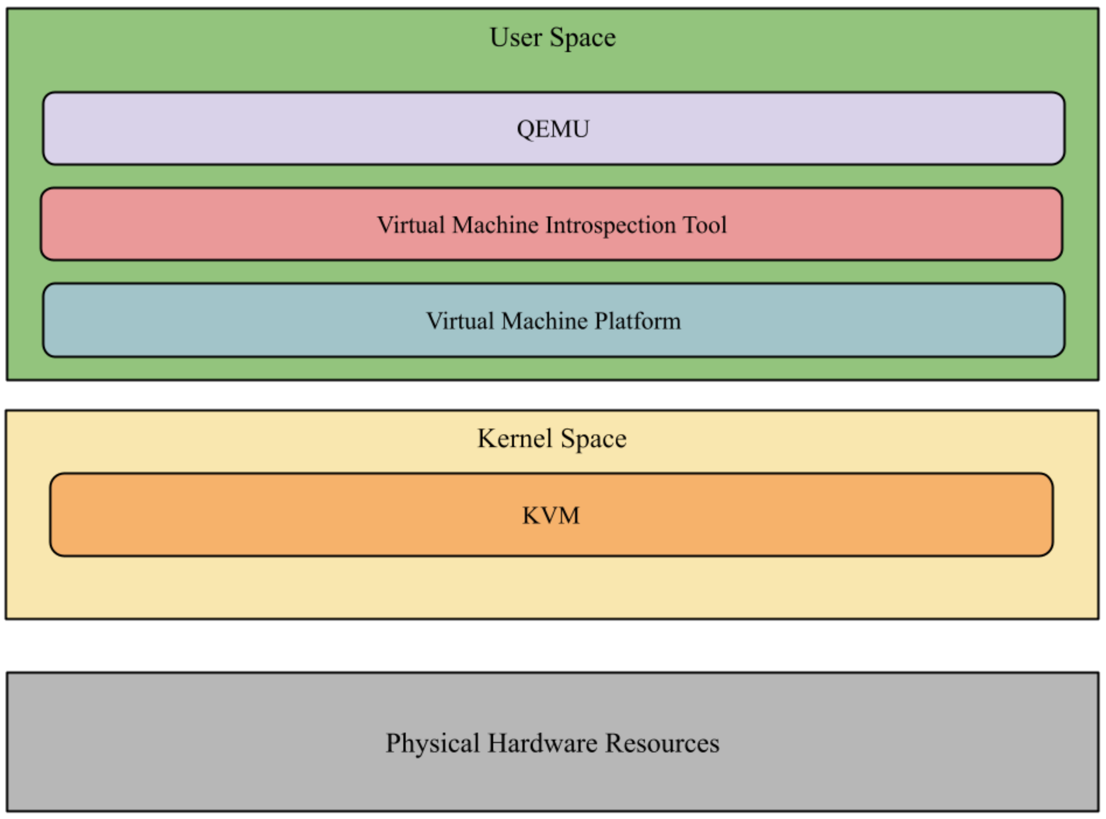

User Space
QEMU
Virtual Machine Introspection Tool
Virtual Machine Platform
Kernel Space
KVM
Physical Hardware Resources

Figure 2.1 Linux architecture with tools described here.

The Virtual Machine Introspection Tool, Virtual Machine Platform, and QEMU are all user space processes as depicted in figure 2.1. Each platform must facilitate communication between its own structures stored in userspace and those stored in kernel space which handle the privileged operations requested by the virtual machines created using these platforms.

KVM is the kernel module which is used by the Virtual Machine Platform to handle these requests from the guest operating system to kernel space. More specifically, KVM informs the kernel when to transfer control to the hypervisor to perform a privileged operation or instruction on behalf of the guest system [7].

3 Related Work

There are many other open source platforms and tools available for virtual machine introspection. Generally, these tools and platforms perform introspection by either pausing the system and gathering information about the memory of the virtual machine or by gathering information while the system is running. Suneja et al. [3] categorize introspection tools using four separate characteristics: "(i) Guest Cooperation,...(ii) Snapshotting,...(iii) Guest Liveness,...(iv) Memory Access Type". The differences between these characteristics are based on the dependencies of the method being used. Suneja et al. [3] state that guest cooperation depends on whether the technique requires a guest operating system to cooperate with the host machine to produce state information. Guest liveness depends on whether the technique used forces the guest to halt before gathering state information. Snapshotting depends on the timing of when the state information is gathered. A technique utilizes snapshotting if it creates a representation of the system at one single point in time rather than gathering data over a period of time. Memory access type depends on "the type of interface provided to the guest space" with the differences between interfaces being "address space remapping, reads on a file descriptor, or through an interface provided by a VM manager" [3]. Many common platforms such as Xen and QEMU provide methods and tools that implement the four separate characteristics uniquely.

One such tool used by QEMU is called the QEMU Monitor [9]. This tool provides the commands `memsave` and `pmemsave` to write the virtual machine's entire memory to a file [3]. These commands are both considered "halt snapshots" which must stop the virtual machine in order to gather a snapshot of the virtual machine state. In order to use these commands the memory map of the virtual machine must be previously known because these commands expect a starting address and a size of memory to dump as input parameters [9]. Without knowledge of the memory layout, the memory dump will be very hard to interpret.

Another tool used to extract state information from a virtual machine is QEMU's Guest Agent [10]. This tool requires guest cooperation to gain access to virtual machine state structures [3]. This tool allows the host system to gather information about the virtual machine as well as control other aspects including virtual processor and guest time configuration [10]. All of these aspects help host hypervisors to keep track of the state of virtual machines being run by QEMU, but this tool must be run within virtual machines. The Virtual Machine Platform does not support commonly used guest agents, but a custom guest agent could be developed to work with the Virtual Machine Platform. The Virtual Machine Introspection Tool fulfills this role as a custom guest agent which can interpret state data from and interact with the Virtual Machine Platform.

4 Project Goals

The main goal of the Virtual Machine Introspection Tool is to provide an accurate representation of the Virtual Machine Platform's state. The ability to introspect is needed to provide an accurate computation history for the Virtual Machine Platform. The structures of the Virtual Machine Platform are not accessible by commonly used tools such as QEMU Guest Agent or other introspection tools. The Virtual Machine Introspection Tool provides access to these structures due to its close integration with the Virtual Machine Platform. By using the Virtual Machine Introspection Tool, state information can be gathered and organized into multiple comma separated value files.

The secondary goal of the Virtual Machine Introspection Tool is to be user-friendly and easy to use. The tool's user interface does this by printing data into tables which makes it convenient for the user to interpret and understand. The tool also caters to the user by allowing them to print out any single index or combination of indices. This allows users to only display data that they find relevant, rather than displaying all of the data all at once.

5 Introspection Tool

5.1 Overview

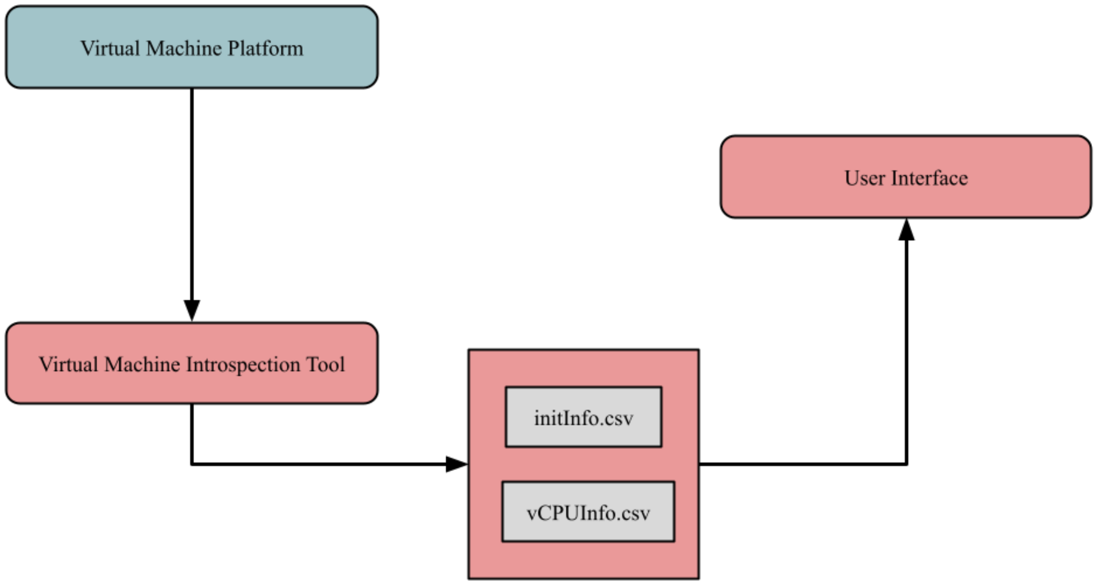


Figure 5.1 Virtual Machine Introspection Tool diagram.

The goal of Virtual Machine Introspection Tool is to generate an accurate representation of the operation of the Virtual Machine Platform as it executes instructions. To do this, the capability to view and inspect the memory of the Virtual Machine Platform needed to be developed. First, an extension to the existing build framework of the Virtual Machine Platform was created. The existing build framework consists of multiple CMake files structured to compile all source code and tests of the Virtual Machine Platform. Next, the capability to monitor the structures within the virtual machine needed to be created. Finally, the Virtual Machine Introspection Tool needed to present the data to the user in a useful and convenient manner.

The main components of the Virtual Machine Introspection Tool are the introspection executable and the user interface as shown in figure 5.1. The introspection executable first gathers data from the Virtual Machine Platform before writing the data to either initInfo.csv or vCPUInfo.csv. Next, the user interface will read the data stored in initInfo.csv and vCPUInfo.csv and display it to the user in the form of a table.

5.2 Extension

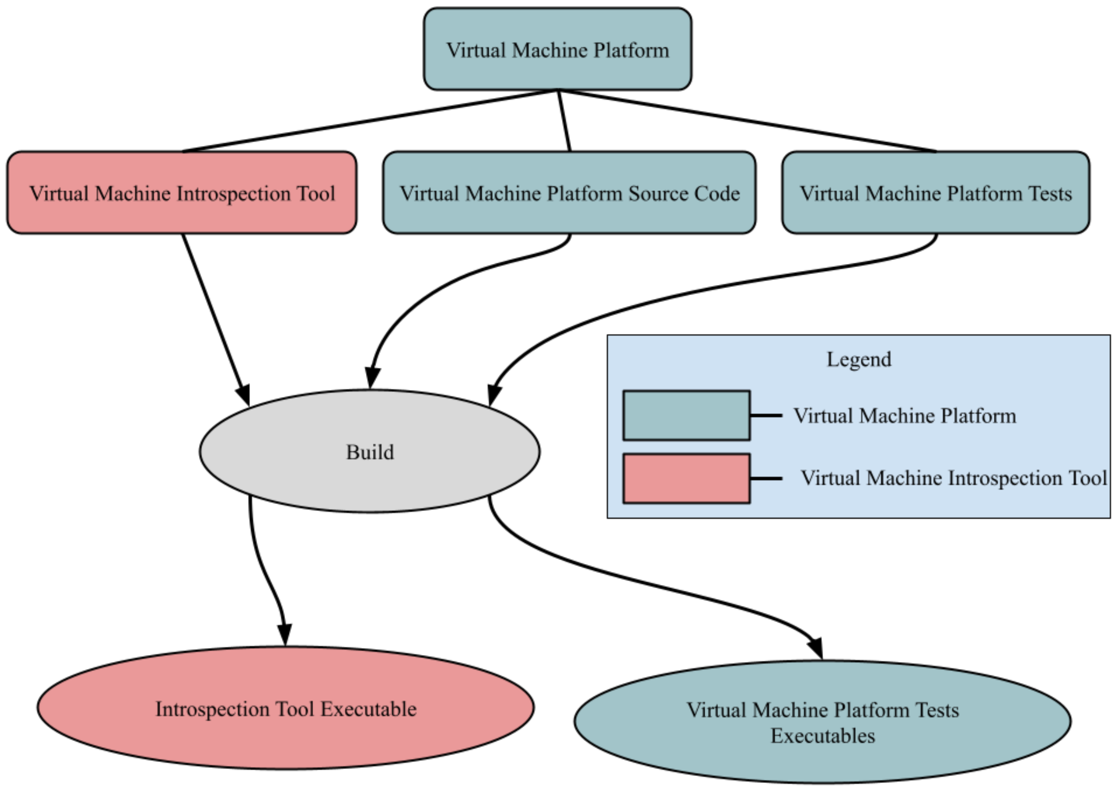

Virtual Machine Platform
Virtual Machine Introspection Tool
Virtual Machine Platform Source Code
Virtual Machine Platform Tests
Legend
Virtual Machine Platform
Virtual Machine Introspection Tool
Build
Introspection Tool Executable
Virtual Machine Platform Tests Executables

Figure 5.2 Extension to the Virtual Machine Platform.

The first step of creating this introspection tool began with adding an extension to the existing Virtual Machine Platform. This existing build framework already contained the Virtual Machine Platform source code and tests as shown in figure 5.2. Adding this extension included editing and adding the proper CMake files to compile within the existing build framework for the project. An executable binary is generated for the introspection tool which allows for ease of execution as well as integration within the existing project structure as shown in figure 5.2. This allows for easy access to all of the structures and functions of the Virtual Machine Platform which are used to create, maintain, and close a virtual machine. The Virtual Machine Introspection Tool keeps track of the following structures within the virtual machine: virtual central processing units (vcpu), vcpu register values, vcpu special register values, virtual machine initialization structures, KVM management structures, and vcpu process exit values. Access to these structures is made available through the Virtual Machine Platform's shared library structure which is created with the CMake files.

The vcpu structure stores information about the state of each virtual central processing unit created by the virtual machine. Within these vcpu structures, there is information stored about the following: identification number, file descriptor, x86 register values, special register values, memory flags, and process exit reason. The identification number is used to uniquely identify each vcpu and differentiate between which vcpu is currently being used or monitored. The file descriptor is the interface between the vcpu and the real cpu that is doing the computation. The x86 register values that are stored are as follows: rax, rbx, rcx, rdx, rbp, rdi, rflags, r8, r9, r10, r11, r12, r13, r14, r15. All register values are stored as 64-bit integers. Special register values are also stored as 64-bit integers and are interpreted as such when collecting their values. These special registers include cr0, cr2, cr3, cr4, cr8, efer, and apic-base. The memory and exit flags of a vcpu are stored as 32 bit integers. The exit reason is also converted from a flag into a readable string before being stored by the Virtual Machine Introspection Tool.

All other information including KVM management structures and virtual machine initialization structures are stored and handled separately from vcpu structures. KVM management structures that are stored include the KVM version as well as the file descriptor used to communicate with KVM. The virtual machine initialization structures that are stored include the number of virtual cpus being used in the virtual machine, the file descriptor of the virtual machine, the memory size used by the virtual machine, and the memory size of the virtual cpus.

5.3 Introspection

The second step required to create the introspection tool was to add the capability and functionality to begin monitoring the virtual machines created by the Virtual Machine Platform. The introspection tool must first instantiate a new virtual machine. Next, a unique identifier is created or provided by the user as a command line argument to determine which log files to write to. This unique identifier is made up of a unique combination of 8 ascii characters. It is then used to create two separate files for storing data gathered during this phase. One file called “initInfo” is used to store all initialization information that is gathered after virtual machine creation and initialization occurs. The other file is called “vCPUInfo” and it stores all information regarding each vcpu’s operation. This includes all information stored in the vcpu structure as described previously. These files will have their unique identifier appended to the filename before they are first created. For example, the filenames with a unique identifier of “64LN0HtO” would be: “initInfo64LN0HtO.csv” and vCPUInfo64LN0HtO.csv”. After this, the virtual machine must be configured to pause after each process completes execution in order to write the data gathered to the log files. The next step is to properly populate the vcpu registers so that the vcpu would then be ready to run a new set of instructions. Next, the Virtual Machine Introspection Tool will attempt to run a program written in x86 assembly on the virtual machine. Finally, after this program completes execution and a return to user space is made from KVM, the tool will store the information from the previously mentioned structures in the two uniquely identified log files.

The use of two uniquely identified log files instead of one was chosen to logically separate the data for future entries using the same unique identifier. Since the introspection tool allows the reuse of unique identifiers, tracking changes to specific virtual machine configurations is possible. During this step, the tool looks within the current directory to find the “logs” folder. If the “logs” folder does not exist within the current operating directory of the introspection tool, the tool will create a new “logs” folder within the current operating directory. If the user provides a unique identifier as the command line argument, the tool searches for this unique identifier within the logs folder and will produce a user error message and exit the tool if the unique identifier is not found. If the unique identifier is found, the tool will append all newly gathered state data to the two uniquely identified files. The reuse of unique identifiers is the main motivation for separating the data into two files as shown in figure 5.3. Initialization data is easier to both track and view when it is in a separate file rather than stored along with the vCPU data in the same file. vCPU data gathered during one run of the executable can create multiple rows of data while initialization data only takes up one row per run of the introspection executable.

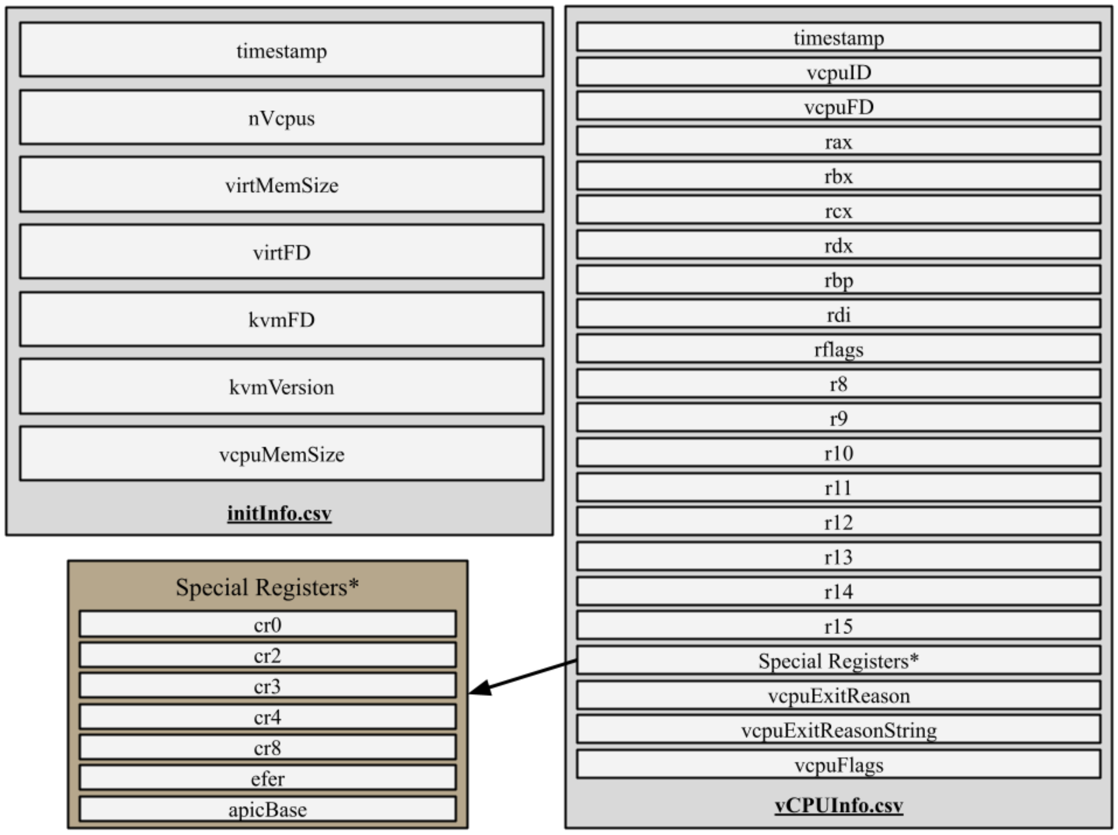


Figure 5.3 Virtual Machine Introspection Tool Executable output.

The Virtual Machine Introspection Tool stores the headers of each column into the respective uniquely identified csv files before storing data gathered via monitoring and introspection. These headers are depicted and split into their respective files within figure 5.3. The headers are stored as the first line of each newly created uniquely identified csv file. With this information, a representation of the current state of the virtual machine can be created. Next, all of this data needed to be presented to the user via the user interface.

5.4 User Interface

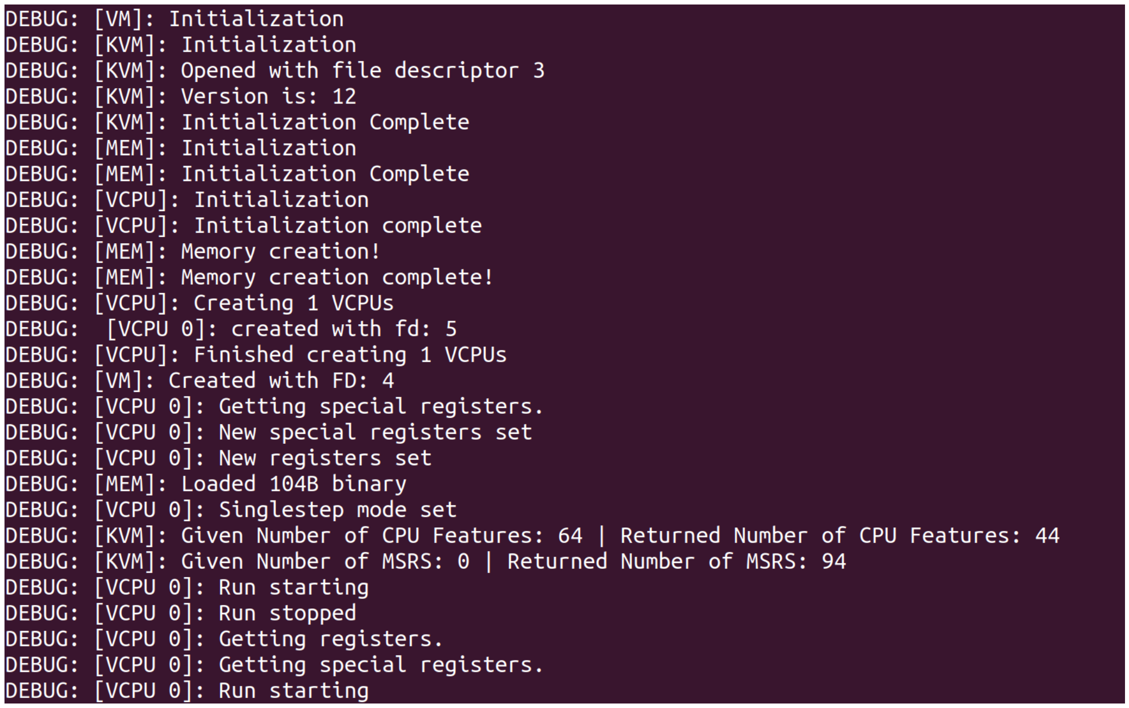
The third step needed to create the introspection tool was to build out a useful and easy to understand user interface. This user interface needed to have the ability to read in data from the csv log files and to also aggregate and organize the data in a presentable manner for the user. To accomplish this, a program was written using Python which reads in csv data by using Python's pandas library csv reading functionality. Two core functions were written to handle user input in an intuitive way.

The `ls` command is used to display the unique identifiers of all log files found within the provided log folder. The user is able direct the interface to their custom log folder by providing the path to the log folder as the command line argument when running the program. Otherwise, the program assumes that the log folder is contained within the current directory that the program is run from. Within these log folders, each uniquely identified pair of log files is read to determine whether the file is compatible with the user interface. If the pair of log files follow proper formatting, the 8 character unique identifier is listed on a new line for the user. Each unique identifier contained within the log folder is then subsequently printed on a new line for the user's convenience when using the `ls` command.

The `print` command is used to display the data contained within the uniquely identified csv log files. This command can also be invoked by just typing `p`, but it expects at least one parameter to be provided by the user. The first expected argument is the unique identifier of the pair of log files to be displayed. If no other arguments are provided, all data stored in the pair of uniquely identified log files is printed to the screen for the user to see in the form of tables. This would include both vcpu information and initialization information displayed as two separate tables. If a second argument is provided, a comma separated list of headers is expected. These headers are the same headers shown in figure 5.3. There are no spaces between separate list items in this comma separated list. If this second argument is provided, only the headers given as arguments will be displayed in the subsequent tables if they exist. Each line of data is timestamped to the time that it was printed so that time of execution can also be tracked easily.

5.5 Operation and Verification

The Virtual Machine Introspection Tool requires a specific set of steps to both run and verify the output produced by this tool. To run the tool, the user must first have access to the Virtual Machine Platform and the Virtual Machine Introspection Tool source code. Next, the user will compile the Virtual Machine Platform and the Virtual Machine Introspection Tool by running `make` within the Virtual Machine Platform project structure. Next, the user can execute the Introspection Tool Executable by navigating to the "/build/bin/" directory within the Virtual Machine Platform project structure and issuing the following command with administrator privileges: `./introspectionTool`.

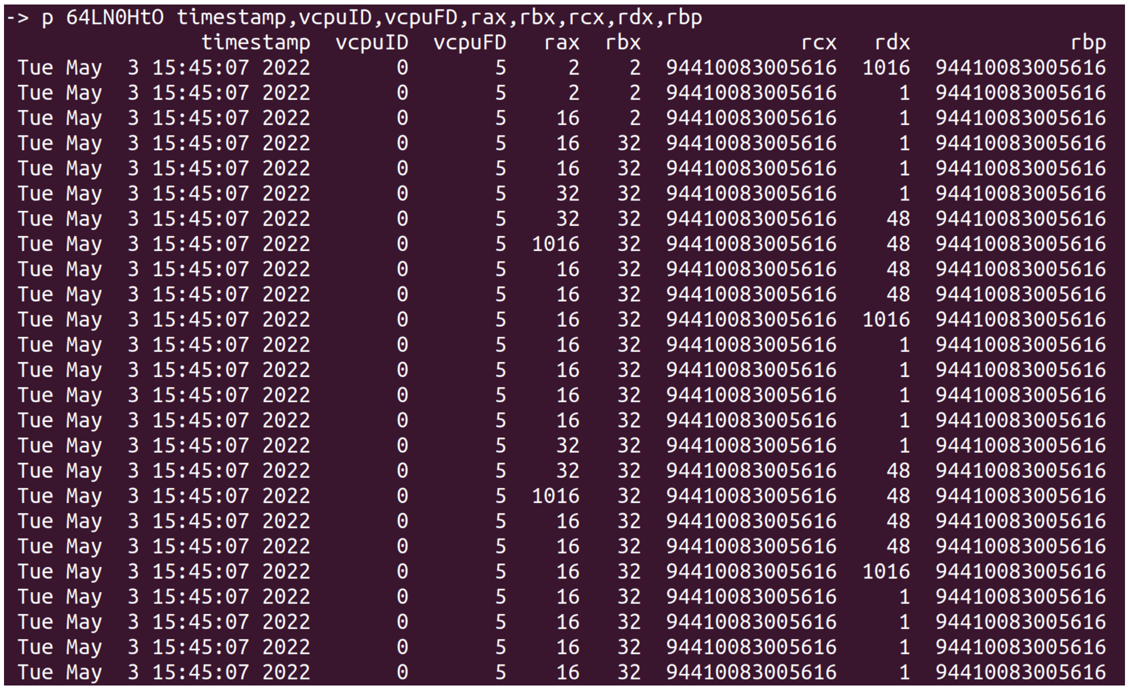


```
DEBUG: [VM]: Initialization
DEBUG: [KVM]: Initialization
DEBUG: [KVM]: Opened with file descriptor 3
DEBUG: [KVM]: Version is: 12
DEBUG: [KVM]: Initialization Complete
DEBUG: [MEM]: Initialization
DEBUG: [MEM]: Initialization Complete
DEBUG: [VCPU]: Initialization
DEBUG: [VCPU]: Initialization complete
DEBUG: [MEM]: Memory creation!
DEBUG: [MEM]: Memory creation complete!
DEBUG: [VCPU]: Creating 1 VCPUs
DEBUG:  [VCPU 0]: created with fd: 5
DEBUG: [VCPU]: Finished creating 1 VCPUs
DEBUG: [VM]: Created with FD: 4
DEBUG: [VCPU 0]: Getting special registers.
DEBUG: [VCPU 0]: New special registers set
DEBUG: [VCPU 0]: New registers set
DEBUG: [MEM]: Loaded 104B binary
DEBUG: [VCPU 0]: Singlestep mode set
DEBUG: [KVM]: Given Number of CPU Features: 64 | Returned Number of CPU Features: 44
DEBUG: [KVM]: Given Number of MSRS: 0 | Returned Number of MSRS: 94
DEBUG: [VCPU 0]: Run starting
DEBUG: [VCPU 0]: Run stopped
DEBUG: [VCPU 0]: Getting registers.
DEBUG: [VCPU 0]: Getting special registers.
DEBUG: [VCPU 0]: Run starting
```

Figure 5.4 Introspection Tool Executable output when run as an administrator.

After execution of the Introspection Tool Executable, the shell displays multiple debug messages from the Virtual Machine Platform with information about the execution of the virtual machines created by Introspection Tool Executable. A sample of these messages are shown in figure 5.4. These messages are printed automatically when using a virtual machine created by the Virtual Machine Platform. The messages are used to verify that a virtual machine ran properly and did not crash unexpectedly. The Virtual Machine Platform will communicate to the user that an issue occurred while running the Introspection Tool Executable using this interface. If no issues occur during the execution of the Introspection Tool Executable, the final message that will be output to the shell will indicate the unique identifier for state data storage. For example, if the unique identifier is"64LN0HtO", the final message will be "Your unique log file identifier is: 64LN0HtO". Next, the Python user interface is used to read and interpret the state data gathered by the Introspection Tool Executable.

To run the user interface, the user must first locate the “logs” directory and the Python file named “searchTool.py” which contains the source code for the user interface. This file is located within the Virtual Machine Introspection Tool source code directory. As previously mentioned the logs directory is created within the directory that the Introspection Tool Executable is run in. Once both are located, the tool can be run by using the following command: `python3 searchTool.py [logs directory path]`.



```
-> p 64LN0HtO timestamp,vcpuID,vcpuFD,rax,rbx,rcx,rdx,rbp
                timestamp  vcpuID  vcpuFD   rax  rbx             rcx   rdx             rbp
 Tue May  3 15:45:07 2022       0       5     2    2  94410083005616  1016  94410083005616
 Tue May  3 15:45:07 2022       0       5     2    2  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16    2  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    32   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    32   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5  1016   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616  1016  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    32   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    32   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5  1016   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616    48  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616  1016  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
 Tue May  3 15:45:07 2022       0       5    16   32  94410083005616     1  94410083005616
```

Figure 5.5 User interface print output with list of headers provided.

Once the user interface is running, it prompts the user for a command. Figure 5.5 shows output of the `print` command using headers from the “vCPUInfo.csv” file. Only a sample of the headers are shown, figure 5.3 displays all of the headers that are available for printing.

Data validation and verification is done by visually comparing the values outputted by the user interface to the expected output from the x86 programs that are run by the virtual machine. These programs are designed to change specific register values within the vcpu. Register values are recorded by the Introspection Tool Executable after each x86 instruction completes execution. Figure 5.5 displays these changes as they were recorded by the Introspection Tool Executable which are shown in the columns containing values for rax, rbx, and rdx. Since the expected register value changes are known before execution, the recorded values can be compared to expected changes. Using this method of comparing recorded values to expected changes, gathered state data can be verified.

6 Future Work

Future extensions to the Virtual Machine Introspection Tool can be made to increase the power and capability of this tool. One way the tool can be built upon is by expanding more ways for the user to interact with the application such as by adding a graphical user interface. The tool can also be expanded by implementing the tool as a part of the shared library structure of the Virtual Machine Platform rather than as a separate introspection executable. This expansion would allow any application using the Virtual Machine Platform to perform introspection and track virtual machine state data. The tool could also be expanded to perform analysis on the log files gathered to gain new insights about the Virtual Machine Platform.

Virtual machine introspection can provide further insights into the operation of a virtual machine. One previous study [2] uses forensic memory analysis tools to analyze a running virtual machine to search for and identify malicious processes running within the virtual machine. One way this study identified these processes when using introspection was by combining the list of running processes with the list of open remote connections on the virtual machine. After malicious network traffic was identified using other analysis tools, they determined which process was sending the traffic using the combined state data, and they denied traffic from that process. Their main goal was to integrate existing forensic memory analysis tools with an existing virtual machine introspection tool. Similar methods of connecting separate pieces of information may be used in the future to be able to deduce more high level information from the operation of virtual machines created using the Virtual Machine Platform.

7 Conclusions

The Virtual Machine Introspection Tool provides exclusive access to the structures stored by the Virtual Machine Platform. Other tools and introspection frameworks are unable to access these structures due to the control and restriction requirements of the platform. Tight integration with the Virtual Machine Platform is needed for interpreting the structures that are contained within virtual machines created using the platform.

The Virtual Machine Introspection Tool helps the Virtual Machine Platform by monitoring and introspecting on the operation of virtual machines on the Virtual Machine Platform. Introspection is an important concept to use since it can reveal inconsistencies or flaws within the operation or state of a virtual machine. Introspection also provides a historical record of the operations performed by the virtual machine during its existence within the host machine. This historical record also includes data about the state of the virtual machine and how this state changes as instructions are executed by the virtual machine.

The Virtual Machine Introspection Tool also provides a command-line based user interface that allows the user to display the historical state and operation data of the virtual machine over time. This interface gives the user flexibility with how the data is displayed and allows the user to pick and choose which data they would like to look at. Overall, this tool was built to add monitoring and introspection capabilities to the Virtual Machine Platform.

8 References

[1] T. Garfinkel, M. Rosenblum, and others, “A virtual machine introspection based architecture for intrusion detection.,” in *Ndss*, 2003, vol. 3, no. 2003, pp. 191–206.

[2] B. Dolan-Gavitt, B. Payne, and W. Lee, “Leveraging forensic tools for virtual machine introspection,” Georgia Institute of Technology, 2011.

[3] S. Suneja, C. Isci, E. de Lara, and V. Bala, “Exploring VM Introspection: Techniques and Trade-Offs,” in *Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, New York, NY, USA, 2015, pp. 133–146. doi: 10.1145/2731186.2731196.

[4] S. N. T. Chiueh and S. Brook, “A survey on virtualization technologies,” *Rpe Rep.*, vol. 142, 2005.

[5] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux virtual machine monitor,” in *Proceedings of the Linux symposium*, 2007, vol. 1, no. 8, pp. 225–230.

[6] M. Lipp *et al.*, “Meltdown: Reading kernel memory from user space,” in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 973–990.

[7] Y. Goto, “Kernel-based virtual machine technology,” *Fujitsu Sci. Tech. J.*, vol. 47, no. 3, pp. 362–368, 2011.

[8] F. Bellard, “QEMU, a fast and portable dynamic translator.,” in *USENIX annual technical conference, FREENIX Track*, 2005, vol. 41, no. 46, pp. 10–5555.

[9] QEMU team, “QEMU Monitor.” QEMU team, 2016. [Online]. Available: https://people.redhat.com/pbonzini/qemu-test-doc/_build/html/topics/pcsys_005fmonitor.html

[10] The QEMU Project Developers, “QEMU Guest Agent.” The QEMU Project Developers, 2022. [Online]. Available: https://qemu-project.gitlab.io/qemu/interop/qemu-ga.html