
External Monitoring of Protocol Stacks with the VMI Toolset

Rheinisch-Westfälische Technische Hochschule Aachen
LuFG Informatik 4 Verteilte Systeme

Diploma Abstract

Marko Ritter

Advisors:

Dipl. Inf. Elias Weingärtner
Prof. Dr.-Ing. Klaus Wehrle

Contents

0.1	Introduction	4
0.2	Problems with the Introspection	5
0.3	Related Work	6
0.4	Concept for the VM Introspection Toolset	7
0.4.1	Monitor kernel level applications	7
0.4.2	Signature scanning	8
0.4.3	Triggering the capture	9
0.5	Evaluation of the VMI Toolset	10
	Bibliography	11

0.1 Introduction

There are several practical applications for the introspection of virtual machines starting from basic debugging to integrity monitoring. Capturing of system states and the possibility to start/stop them with almost no overhead makes them the perfect debugging environment. The introspection of system states or even a running VM is however not so easy. This work should help ease the process of debugging a kernel level application/module by providing an easy way to read the content of marked variables from outside the VM at any time.

By now the process of configuring an introspection tool to fit one kind of virtual machine included several manual steps and separate tools. This is hopefully going to change. This toolset will automate most of this steps so the user will just have to mark the target variable inside the code and can then read the actual contents in a safe place outside of the virtual machine.

This work is build upon the work from Christoph Terwelp which gave a prove of concept for basic VM Introspection with XEN OS.

0.2 Problems with the Introspection

The main challenges at hand are to find the right memory address in the virtual machine's memory block and to interpret it right. In other words, you have to find the position of the variable you want to look in and you have to know the correct type of the variable in order to get a meaningful debug output.

The problem in finding the correct memory address inside the virtual machine results from the host system not knowing where the guest system is loading the process in its memory space. In fact the host doesn't know anything about the guests memory except its size and where it is accessed and how. It can only make good assumptions based upon the type of the guests operating system. Using an open source OS enables us to take a look at the source and reconstruct the memory loading process. In this case we have a chance to interpret the memory right and maybe find what we searched for straightforward.

The only other way to find something in a guests memory, which is using a closed source or unknown operating system, is to bruteforcely scan the whole memory for a special signature we place around our target zone beforehand. Assuming a static kernel memory structure this may even enable us to reverse engineer other OS structures later.

Another problem is to know when to capture or read the virtual machine's memory. One approach for this is to just capture the variable contents at predefined times (logging). Another could be to trigger the capture through events from inside or outside the VM. For example triggers in the network stack which could fire when a packet arrives. Triggering from the inside of a VM supposes a quick connection between the capturing module in the Host and the triggering module inside the Guest System. More on this later in the concept idea.

0.3 Related Work

I can only endorse “The Definitve Guide to the Xen Hypervisor” [2] for a good introduction and overview for Xen. Further details on Virtualization with Xen can be found in an article from a Linux Symposium [1].

There are several existing works on VM Introspection. An introduction to the current ProMox Project exists with “Promox: A protocol stack monitoring framework” [3].

The most works on introspection use it for security monitoring. Examples for this are [4] and [6]. Another work [5] introduces VIX (an introspection toolsuite) used for integrity and security monitoring of VMs.

[Ich werde die noch ein bißchen ausführlicher erläutern.]

0.4 Concept for the VM Introspection Toolset

0.4.1 Monitor kernel level applications

To get a quick overview of the preliminary concept Figure 1 roughly sketches the kernel level introspection process with the help of the toolset.

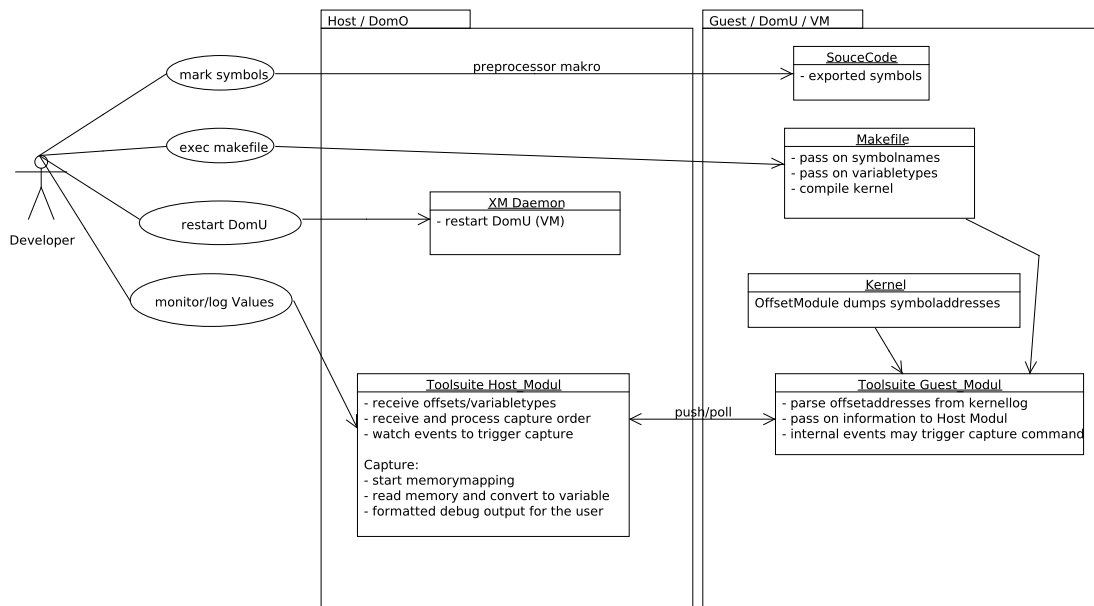


Figure 1 using the toolset to monitor a kernel level application

The left side shows the usecases the developer and in this case the user of the toolset has to perform in this process. The right side shows host and guest system and the different modules inside them. The arrows are linking the usecases to the according modules where the actions are performed in.

The first step in monitoring a specific variable is to decide on and mark one in the code. Therefore a C preprocessing makro will be available for the programmer. The buildsystem will then compile the code and extract the information needed to track the variable. In more detail: The variable will be exported as a symbol and the exact type of the variable will be saved for later interpretation of the raw memory data. For this function several C and makefile makros will have to be build. The data handling will be done by the toolset's guest module which will be called by the buildsystem and transmit the information to the host module later on. The additional tasks of the buildsystem and how it communicates with the guest module are sketched in Figure 2.

Being a kernel level application, the virtual machine and its kernel have to be restarted to run it. As shown in Figure 3, the guest module of the toolset will use a special kernel module to dump out the symbol offset for the variable we are trying to watch. This offset has to be transmitted to the host module along with the variable type definition. The transmission can be realized using different technologies. One

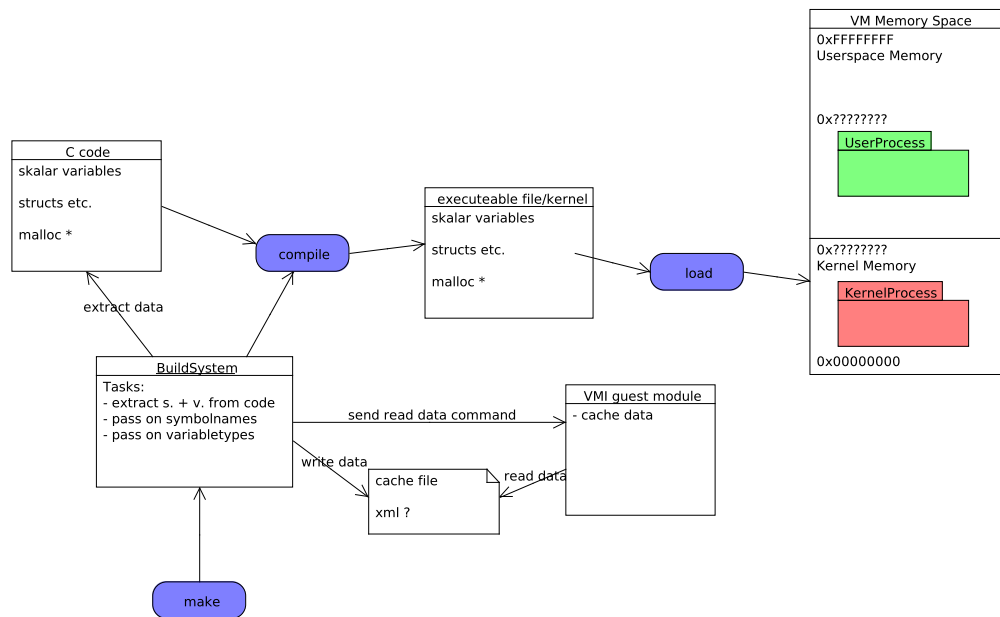


Figure 2 tasks of the buildsystem

could use an old fashioned socket based client/server model to provide the information to the host module. We could also use some fast Xen specific communication ways between host and guest system. As there are shared memory pages, memory page transfer or even just the xen console which could be parsed by the host module.

Assuming the host module now knows the absolute memory offset and type of the variable, it is able to just map the memory into a variable of the same type in the userspace of the host system. The host module can then format a debug output for the user and present it to him.

0.4.2 Signature scanning

So much for kernel level applications. If we want to monitor userspace application, we have to revert to signature scanning inside the process memory area or even worse the virtual machine's whole memory block. While being able to find the absolute offset of the process memory area through traversing the kernel structures and the process list of the guest operating system, assumed we are working with a guest OS we have detailed kernel information on. We cannot preparatory calculate the relative offset of the variable inside the process memory area. So building a special signature around the target variable should help finding it in the memory using signature scanning.

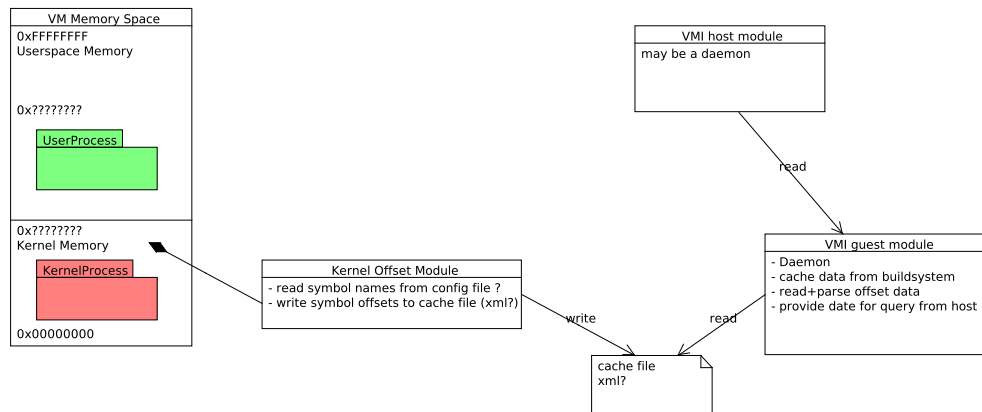


Figure 3 tasks of guest module and kernel offset module

0.4.3 Triggering the capture

The simplest way to trigger the capture is with a time interval for simple logging. A maybe more useful approach may be to trigger the capture based on certain events in the guest or host system. Since the capture will be carried out by the host module, the guest module must quickly forward any capturing trigger it receives to it. The host module could then pause the VM to take a snapshot and continue it afterwards.

A useful application scenario for a triggered capturing solution would be network packet capturing. A trigger could be implemented into the split driver for the network device (either on guest or host side of the driver), so that it fires when an incoming packet is received in the buffer.

0.5 Evaluation of the VMI Toolset

To show the resulting easement of the introspection process, we plan to evaluate the toolset on a well know kernel level protocol. We will compare the steps needed to perform the introspection with and without the toolkit.

A good example protocol would be TCP/IP on an open source linux guest. We will then mark one target struct inside the kernel tcp/ip implementation and perform the whole introspection procedure. The evaluation goal is to monitor and log the struct's values from the host OS.

[Fehlt noch sowas wie ne Schlußbemerkung oder so?]

Bibliography

- [1] Paul Barham, Boris Dragovic, Keir Fraser, Steven H, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt und Andrew Warfield. Abstract xen and the art of virtualization.
- [2] David Chisnall. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, 2007.
- [3] Christoph Terwelp Elias Weingartner und Klaus Wehrle. Promox: A protocol stack monitoring framework. Technical report, Distributed Systems Group RWTH Aachen University Aachen, Germany, 2009.
- [4] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum und Dan Boneh. Terra: a virtual machine-based platform for trusted computing. *SIGOPS Oper. Syst. Rev.*, 37(5):193–206, 2003.
- [5] M. Hay B. Alaska Univ. Fairbanks AK Nance, K. Bishop. Virtual machine introspection: Observation or interference? 2008. URL: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4639020.
- [6] Mendel Rosenblum Tal Garfinkel. A virtual machine introspection based architecture for intrusion detection. 2002. URL: <http://suif.stanford.edu/papers/vmi-ndss03.pdf>.