BYTE ATLAS

05.02.2017

Knowledge Fragment: Hardening Win7 x64 on VirtualBox for Malware Analysis

After some abstinence, I thought it might be a good idea to write something again. The perfect occasion came yesterday when I decided to build myself a new VM base image to be used for future malware analysis.

In that sense, this post is not immediately a tutorial for setting up a hardened virtual machine as there are so many other great resources for this already (see VM Creation). Maybe there is a good hint or two for you readers in here but it's mostly a write-up driven by my personal experience.
The main idea of this post is to outline some pitfalls I ran into yesterday, when relying on said resources. To have others avoid the same mistakes, I hope this post will fulfil its putpose.
In total I spent about 5 hours, 2 hours for setup and probably another 3 hours for testing but more about that later. This could have easily been only one hour or less if I knew everything I'll write down here beforehand. So here you go. :)

The remainder of this post is structured as follows:

1) Goals

Before starting out, it's good to know and plan where we are heading.

My Needs: I'm mostly interested in doing some rapid unpacking/dumping to feed my static analysis toolchain and then occasional do some debugging of malware to speed up my reasoning of selected code areas.
For this, I wanted a base VM image that is able to run as much malware natively as possible, without me having to worry about Anti-Analysis methods.
Potentially, I want to deploy this image later as well for automation.
I don't aim for a perfect solution (perfection is the enemy of efficiency) but a reasonably good one.

OS choice: Windows 7 is still the most popular OS it seems, but since 64bit malware is getting more popular, we should take that into concern as well. So I go with Win7 x64 SP1 as base operating system.

Why not Win10: Well, I want a convenient way to disable ASLR and NX globbaly to allow my malware&exploits to flourish. Since I don't know if it's as easy in Win10 as it is in Win7, I stick with what I know for now.

2) Preparation

In the back of my head, I had some resources I wanted to use whenever I would have to create a new base VM, namely:

1) VMCloak by alier_t
2) VBoxHardenedLoader by hfiref0x (and kernelmode thread as installation guide)
3) antivmdetection by nsmfoo (and blog posts 1 2 3 4 5)

Since I wanted to understand all the steps, I took VMCloak only for theoretical background. VBoxHardenedLoader is targeting a Win7 x64 as host system, however I use **Ubuntu 16.04** with **VirtualBox 5.0.24** so this wasn't immediately usable as well. But it's another excellent theoretical background resource.

Ultimately I ended up using antivmdetection as base for my endeavour.
Since I trial&error'd myself through the usage (in retrospect: I should do more RTFM and less fanatic doing), here's a summary of things you want to do before starting:

1) Download VolumeID (for x64)
2) Download DevManView (for x64)
3) # apt-get install acpidump (used by Python Script to fetch your system's parameters)
4) # apt-get install libcdio-utils (contains "cd-drive", used to read these parames)
5) # apt-get install python-dmidecode (the pip-version of dmidecode is incompatible and useless for our purpose, so fetch the right one)
6) $ git clone https://github.com/nsmfoo/antivmdetection.git
7) $ cd antivmdetection :)
8) $ echo "some-username" > user.ist (with your desired in-VM username(s))
9) $ echo "some-computername" > computer.ist

Okay, we are ready to go now.

3) VM Creation

First, I simply created a new empty Win7 x64 VM.
I used the following specs:

* CPU: 2 cores
* RAM: 4 GB
* HDD: 128 GB
* VT-x activated (needed for x64)
* GPU: 64 MB RAM (no acceleration)
* NIC: Ix Host-Only adapter (we don't want Internet connectivity right away or Windows may develop the idea of updating itself)

Important: Before mounting the Windows ISO, now is the time to use antivmdetection.py.

It will create 2 shell scripts:
1) <DmiSystemProduct>.sh <- Script to be used from outside the VM
2) <DmiSystemProduct>.ps1 <- Script to be used from inside the VM post installation

Run Antivmdetection (outside VM): For me <DmiSystemProduct> resulted in "AllSeries" because I run an ASUS board.
Okay, next step: execute <DmiSystemProduct>.sh - For me, this immediately resulted in a VM I could not start. Responsible for this were the 3 entries
1) DmiBIOSVersion
2) DmiBoardAssetTag
3) DmiBoardLocInChass
Which were set by <DmiSystemProduct>.sh to an integer value and VirtualBox was pretty unhappy with that fact, expecting a string. Changing these to random strings fixed the issue though. So this may be one of the pitfalls you may run into when using the tool. Setting the ACPI CustomTable however worked fine.

4) Windows Installation

Historically: Throw in the ISO, boot up, and go make yourself a coffee.
I had less than 10 minutes for this though.

5) Post Installation Hardening and Configuration

Now we have a fresh Windows 7 installation, time to mess it up.

Windows Configuration: Here are some steps to consider that may depend on personal taste.
1) Deactivate Windows Defender - Yes. Because. Malware.
2) Deactivate Windows Updates - We want to keep our system DLL versions fixed to be able to statically infer imported APIs later on.
3) Deactivate ASLR - We don't want our system DLL import addresses randomized later on. Basically, just create the following registry key (Credit to Ulbright's Blog):

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management] - "MoveImages"=dword:00000000
4) Deactivate NX - whatever may help our malware to run... Basically, just run this in Windows command line (again Credit to Ulbright's Blog):
bcdedit.exe /set {current} nx AlwaysOff
5) Allow execution of powershell scripts - Enter a powershell and run:
> Set-ExecutionPolicy Unrestricted

Run Antivmdetection (in VM): Now we are good to execute the second script <DmiSystemProduct>.ps1.
Some of its benefits:
* ensure our registry looks clean
* remove the VirtualBox VGA device
* modify our ProductKey and VolumeID
* change the user and computer name
* create and delete a bunch of random files to make the system appear more "used".
* associate media files with Windows Media Player
* clean itself up and reboot.

I fiddled a bit with the powershell script to customize it further. Also, after reboot, I removed the file manipulation and reboot code itself to be able to run it whenever I need to after deploying my VM to new environments (additionally, this reduces the runtime from several minutes to <5sec).

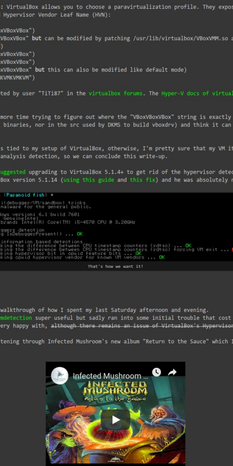
Dependencies: Because malware and packers often require Visual C and NET runtimes, we install them as well. I used:
* MSVCRT 2005 x86 and x64
* MSVCRT 2008 x86 and x64
* MSVCRT 2010 x86 and x64
* MSVCRT 2012 x86 and x64
* MSVCRT 2013 x86 and x64
* MSVCRT 2015 x86 and x64
* MS-NET 4.5.2

Snapshot time! I decided to pack my VM now into an OVA to archive it and make it available for future use.

Now feel free to inflict further harm to your fresh VM.
Installing MS Office, Adobe Acrobat, Flash, Chrome, Firefox all come to mind.

Certainly DO NOT install VBoxGuestAdditions. The only benefits are better adaption of screen resolution and easy shared folders. For shared folders you can also just check out impacket's smbserver.py which gives you about the same utility with a one-liner from your host shell.

PAFish looking good:



6) VirtualBox Hypervisor Detectability

This is no longer an issue when updating to VirtualBox version 5.1.4+, read below.

As initially mentioned, I spent another 3 hours with optimization and trying to get rid of the hypervisor detection.

Note that modifying the HostCPUID via VBoxManage does not fix the identity of VirtualBox which I basically learned the hard way.

Paravirtualization settings: VirtualBox allows you to choose a paravirtualization profile. They expose different combinations of hypervisor bit (HVB) and Hypervisor Vendor Leaf Name (HVN):

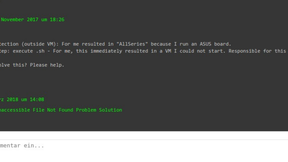
1) none   (HVB=0, HVN="VBoxVBoxVBox")
2) default (HVB=1 HVN="VBoxVBoxVBox" **but** can be modified by patching /usr/lib/virtualbox/VBoxVMM.so as shown above, where we have "**vbvbvbvbvbvb**" instead)
3) legacy (HVB=0, HVN="VBoxVBoxVBox")
4) minimal (HVB=1, HVN="VBoxVBoxVBox")
5) Hyper-V (HVB=0, HVN="VBoxVBoxVBox" but this can also be modified like default mode)
6) KVM   (HVB=0, HVN="KVMKVMKVMKVM")

This was also previously noted by user "TiTi87" in the virtualbox forum. The Hyper-V docs of virtualbox sadly could not help me either.

I will probably spent some more time trying to figure out where the "VBoxVBoxVBox" string is exactly coming from (could not find it in other virtualbox binaries, nor in the src used by DKMS to build vboxdrv) and think it can be ultimately binary patched as well.

However, the issue itself is tied to my setup of VirtualBox, otherwise, I'm pretty sure that my VM itself is looking rather solid now in terms of anti-analysis detection, so we can conclude this write-up.

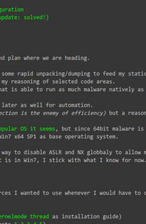
UPDATE 2017-02-06: nsmfoo suggested upgrading to VirtualBox 5.1.4+ to get rid of the hypervisor detection. So I took his advice, moved up to VirtualBox version 5.1.14 (using this guide and this fix) and he was absolutely right:



7) Summary

This post ended up being a walkthrough of how I spent my last Saturday afternoon and evening.
I found nsmfoo's tool antivmdetection super useful but sadly ran into some initial trouble that cost me some time. Ultimately I ended up with a VM I am very happy with, although there remains an issue of VirtualBox's Hypervisor Identification.

I wrote this post while listening through Infected Mushroom's new album "Return to the Sauce" which I can also heavily recommend. :)



08:34

Labels: anti-analysis, hardening, malware, reverse engineering, ubuntu, vbox, virtualbox

Kommentare:

Allan NG 14. November 2017 um 18:26

Hi,

Run Antivmdetection (outside VM): For me resulted in "AllSeries" because I run an ASUS board.
Okay, next step: execute .sh - For me, this immediately resulted in a VM I could not start. Responsible for this were the 3 entries.

How to you solve this? Please help.

Antworten

Vinay 27. März 2018 um 14:08

VirtualBox Inaccessible File Not Found Problem Solution

Antworten

Gib einen Kommentar ein...

Kommentar schreiben als: almassahmed[▾]
Abmelden
Veröffentlichen | Vorschau
Ich möchte Benachrichtigungen erhalten

Neuerer Post | Startseite | Älterer Post
Abonnieren Kommentare zum Post (Atom)

Über mich
ByteAtlas
Mein Profil
vollständig anzeigen