Analysing malware samples

Pablo Collado Soto

**Setting up**

Before setting off on a quest to find malware lurking around, we need to define the scenario we’ll be working on. As stated in the coursework specification, our environment will be a virtual machine running Windows XP.

In order to make it feel like at home we have taken a series of steps so that all the necessary components and tools were available. We’ll briefly describe the entire process so that what we include in this report is as reproducible as possible.

Please note that in the ensuing discussion we’ll refer to the machine running Windows XP as the guest machine and the local machine as the host machine.

**Importing the VM**

The guest machine we are to work with is tremendously specific and so is very hard to exactly replicate. That’s why the way it’s been distributed is through a virtual machine image using the *\*.ova* format. As we are working with Oracle’s VirtualBox virtualization software we just need to open it up and import it as shown in the following screenshot:

**TODO: Get screenshot!**

**Installing the Guest Additions**

We shan’t forget Windows XP uses a graphical desktop environment. We’ll need to make the guest machine’s desktop as big as ours so that we can comfortably move around. To this end, we can install the so called *VirtualBox Guest Additions*which just bundle up a set of drivers designed to make using VMs a seamless experience. Installing it is as easy as going into the *Device* menu within the running VM and clicking on *Insert Guest Additions CD Image…* We can then navigate to Windows XP’s *My Computer* from the *Start* menu and double click on the disk unit to bootstrap the installation. After rebooting the VM we should be ready to go.

It’s likely that before the guest additions can be inserted into the machine we need to create an optical disk bay. We can do so from the settings of the VM before booting it up by just adding a new entry to the *IDE Controller*. We are attaching a screenshot showing how to accomplish that.

**TODO: Get screenshot!**

**Configuring the Network**

As our intention is to “detonate” malware within the machine we should mind the network configuration so that we don’t let any of the samples roam freely on the network. This can be easily accomplished by just clicking on the *Devices > Network > Connect Network Adapter* option within the running VM which will just toggle the NIC (Network Interface Card) as we please. This let’s us have an entirely functional network setup that can be enabled at will. As our host machine is a MacBook Pro we’ll use a *bridged adapter* configuration which will expose the guest VM as an independent machine to the LAN our host machine is connected to. This implies it’ll get it’s very own IP address within the LAN our host is attached to. We can also point out that the network topology we are logically implementing is a layer 2 bridge connecting the host and guest machines so that both still belong to the same network segment (i.e. LAN). If we were to have a Windows-based host we would go with a *NAT* configuration option as that’s less error prone and given the Windows XP machine is not to behave as a server we won’t perceive any functional difference.

Either way, as we intend on leveraging *WireShark’s* capabilities for capturing network traffic we need to install the suitable NIC driver. This will also force us to select the correct NIC when configuring the VM, thing we can do under the *Advanced* section within the *Network* section of the configuration. Given the driver we were provided we’ll choose the *Intel Pro/1000 MT Desktop (82540EM)* NIC so that everything remains compatible. The driver installer can be found in our course repository as usual. Please refer to the annex on the repository structure. Installation is just a matter of double-clicking the executable and accepting the different steps.

**Showing the hidden files**

As more recent Windows releases, we see that Windows XP already made use of the *NTFS* file system for its drives. It, as many others, has support for hidden files. These files are nonetheless not as hidden as we might expect. We can inspect them in a fairly easy way both through the graphical file explorer and from Windows’ CLI (*cmd*). In order to show hidden files on the file explorer we just need to go to the *Control* *Panel* (which we can do from the *Start* menu)and click on the *Folder Options* option. We are attaching a screenshot showing the option we are to tick.

**TODO: Add screenshot.**

We can also use good ol’ *cmd* to navigate the file hierarchy and inspect hidden files. As we know we’ll find a gold mine on *C:\WINDOWS\msagent\intl\MS\_PMAL\_Agent* we might as well show how we can list that hidden directory. We just need to navigate to it’s parent directory with cd *C:\WINDOWS\msagent\intl* to then run *dir /AH.* As seen in *dir’s* help which we can read through with *help dir*, we can control its output through a series of options. Hidden files and folders won’t be shown by default but we can force it to display them with the */AH* which only displays files with the *Hidden* attribute. Then, we can indeed see the *MS\_PMAL\_*Agent directory if we run *dir /*AH after the previous cd statement.

Either way, we need to be aware of the existence of hidden files and take measures so as to have them present and take them always into account.

**Stopping automatic updates**

The malware samples we have been provided are guaranteed to behave as expected if and only if we maintain the correct OS version. That’s why we need to prevent automatic updates from happening. We can easily do so by going to the *Control Panel* and clicking on *Automatic Updates*. We can them disable them form the next window that’ll pop up and that we are including in below.

**TODO: Add screenshot!**

This will guarantee nothing is meddled with as we carry out our analysis so that we can worry only about crucial and important aspects whilst being sure no “funny” behaviours are taking place behind the scenes.

**Regarding the firewall**

As before proceeding to the analysis we can’t be sure of there being any malicious use of the network we decided that shutting the firewall down before having any leads as to what the behaviour of the malware was a little bit of an overkill. We don’t depend on the network for using our tools either so there is no immediate need for doing so. We’ll keep in mind that it’s actually up in case it becomes something to consider at some point. Nonetheless, we can find the configuration under *Control Panel > Windows Firewall*. We are attaching a screenshot showing our current configuration.

**TODO: Add screenshot!**

Please note that we intend to capture network traffic from within the guest itself so there is no need to have any external machines capturing traffic. Should that need arise we’ll shut the firewall down as needed. Again, we don’t want to take any actions that might turn up being unnecessary as we strive to carry out an analysis that’s as little invasive as we can.

**Communicating with the outside world: shared folders**

In order to speed our work process and in an effort to let us work as much as possible from our host machine we have decided to leverage the power of shared folders to share files and folders between the host and guest seamlessly. In order to do so we just need to go to *Devices > Shared Folders > Shared Folder Settings…* on the running VM. We’ll then add a new folder by clicking on the marked icon.

**TODO: Add screenshot!**

We just need to point it to the folder we want to share from our host machine and select the *Auto-mount* and *Make Permanent* options so that it’s added to the VM whenever we boot it up. We’ll see that the shared folder shows up as a networked drive within *My Computer* in Windows XP and that anything we copy to it on one end will appear on the other and vice versa. This will let us move files from one machine to the other in no time thus speeding up the analysis process by a great amount.

**Getting the tools**

A malware analyst is only as good as his/her tools so we need to get our hands on a few handy programs that will aide us in our analysis. We mustn’t fall into the trap of thinking that he more tools we employ the better and more thorough our analysis will be. Whilst it is indeed true that different tools may provide different insight, we should always trust our own judgement as the tools are submitted to us and not the opposite. Thus, once we have concluding evidence on a piece of malware, we won’t continue running it through several tools for the sake of doing it if we believe this is not necessary. That’ll let us employ less tools in a more efficient way so that the overall outcome of our work is as solid as we want it to be. We are including a complete relation of them as well as a short description so that the reader can have an idea of what we are looking for when we use each of them. Note the ones marked with \* were installed by us and their setups can be found within the aforementioned repository. The rest were already present in the imported VM.

**TODO: Add the list at the end! Tentative list:**

**Wireshark**

**InetSim**

**IDAPro**

**Cutter**

**OllyDbg**

**Process Explorer**

**Process Monitor**

**TCPView**

**RegShot**

**PEview**

**PEiD**

**RDG Packte Detector**

**TODO: Explain what other VM we are using for analysing stuff. We are just using or Mac…**

**Better safe than sorry**

Playing around with malware can be risky. We shouldn’t forget that these pieces of code are designed to, in the least, access sensitive and in the worst to wreck the machine they run on. This implies we might get to a point our VM is “broken” beyond repair so we had better get a *snapshot* of the machine before proceeding so that we have a correctly configured fallback point that we can always fall back to. We can easily get this snapshot by hitting *Cmd + T* on macOS and naming it to something suitable such as *“Backup State”*. We can also opt to add a description to it but we believe it’s not necessary given the title.

Once saved we can always start the machine from the saved state from the VirtualBox manager as seen in the following picture.

**TODO: Add screenshot!**

**Final notes**

In order to run the VM we just need to launch it from within *VirtualBox’s* manager. The user-password pair is *Administrator-AVictim.*

With that we are ready to get our hands dirty. We’ll include the analysed malware samples as we found them so that the chronology becomes another valuable asset showing how the analysis progressed with time.

**The first malware sample**

Malware will want to run undetected as long as it possibly can. This should make us consider the different ways programs can be started to try and see whether they are being weaponized to launch malicious software.

One such example are services. As seen on page 152 of [PMAL] these services are not run as standalone process. Instead, the *Windows Service Manager* will schedule them for execution according to their configuration. The tasks run in this way will remain in the background in the same way *daemons* do. The term *daemon* refers to programs that are not designed for the user to interact with. They carry out a task such as providing an HTTP server accepting external connections and their behaviour can be configured through text-based configuration files in most cases but not by direct interaction.

In order to inspect the existing services on a machine we can just *Right-click* on *My Computer* and select the *Manage* option. We can then navigate through *Services and Applications > Services* to find a list of available services. If we look at the names and descriptions we’ll soon stumble upon the *GrayPigeon\_Hacker.com.cn* service with description ª“∏Î◊”∑˛ŒÒ∂À≥Ã–Ú°£‘∂≥Ãº‡øÿπ‹¿Ì.

**TODO: Add services list screenshot!**

It certainly doesn’t look like a legitimate entry so we can click on it to gather more information about it. In doing so we’ll see that it is launching the *C:\WINDOWS\Hacker.com.cn.exe* file when it’s invoked so a sensible next step would be to analyze said file.

We’ll also try to gather some more information about the service itself to see whether it’s being started automatically upon boot, what type of service it is… This type of information can be found on the *Windows Registry* which we can access by going to *Start > Run* and typing *regedit*. We then have to navigate to *HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services* and look for our service. Please note we’ll contract *HKEY\_LOCAL\_MACHINE* to *HKLM* from now on. Once we get to *HKLM\SYSTEM\CurrentControlSet\Services\GrayPigeon\_Hacker.com.cn* we’ll see the different entries configuring this service.

**TODO: Add registry screenshot!**

They are quite unreadable, so we believe it’s useful to discuss the *sc* command. As seen in its help (which we can access with *sc /?*) it’s a program that communicates with the *NT Service Controller* *and Services* so that it can get information about services. We’ll be looking into the service’s configuration, so we’ll want to be using the *qc* option. Then, by running *sc qc “GrayPigeon\_Hacker.com.cn”* we’ll get the same information as before but in a more comfortable format.

**TODO: Add sc screenshot!**

We see how the service is set to be run automatically at boot (its *START\_TYPE* is *AUTO\_START. See* [*this*](https://docs.microsoft.com/en-us/windows/win32/services/automatically-starting-services) *link for more information*) and it’s of the *WIN32\_OWN\_PROCESS* type which means it’s code is an *EXE* file as stated in page 153 of [PMAL], something we already knew before!

1. **Where is the malware located?**

Either way, we have found that **the malware itself is contained in the aforementioned *EXE* file on *C:\WINDOWS\Hacker.com.cn.exe****.* We’ll try to copy it to our shared folder so that we can analyze it on our host machine which is way faster than the VM so that scrolling and combing the file is not that cumbersome.

Even though one might think copying the file would have been easy we had to fiddle with the settings a bit. It turns out the *Hacker.com.cn.exe* was marked as a *System File* which made it dodge our current folder options. If we navigate to *C:\WINDOWS* through a command line we can indeed run *dir /AH* once more and we’ll eventually see the *Hacker.com.cn.exe* appear as it should. If we run *attrib* on it to see its attributes we’ll see the following output:

**TODO: Add attrib screenshot**

The *SHR* output indicates it’s a *System, Hidden and Read-only* file. This can be seen on attrib’s help with *help* attrib. If we visit the folder view options once more we’ll have to uncheck the option hiding system files as seen in the following image.

**TODO: Add new folder config**

In doing so the file will show up in the graphical explorer as well. We just want to point out that we could have copied the file to our shared folder with the *copy* command without a problem and we could have skipped tweaking the folder settings again. This shows the power of text-based interfaces.

If we try to copy the file we’ll stumble upon the following error:

**TODO: Add error screenshot**

Nonetheless, the service that launched the file is already stopped, that is, it’s already stopped as we could see in some of the screenshots above. Then we’ll just disable it so that it doesn’t start at boot and try to copy the file once it’s not run in the first place. Once we did that the file copied without a problem. Note that if we are to use the *copy* command to copy the file over, we need to run *attrib -s -h C:\WINDOWS\Hacker.com.cn.exe* beforehand so that it stops being a system and hidden file. We can then run *copy C:\WINDOWS\Hacker.com.cn.exe Z:\MW\_sample\_1* to create the *MW\_sample\_1* in our shared folder.

Now we can run the *file* command on the sample from our *macOS* machine to find the following output:

**TODO: Add file type screenshot.**

Taking this information into account we can be sure the file is indeed a **32-bit Portable Executable file**. Now, this *PE* format is the

As seen in page 14 of [PMAL] this file format is the one employed in Windows executables, *DLLs* (Dynamic Link Libraries) and object code (the one we get when running the compiler on source code before passing it through the linker). It basically tells the Windows program loader all the information it needs for loading and running the program and it also contains the executable compiled program code itself. Information included in this header are the needed library functions, and space/memory requirements among others. As this information is provided in a known format, we can easily spot this type of file.

We can now try to inspect the file using *PEviewer* so that we can clearly inspect the different sections composing the file.

**TODO: Add PEview screenshot**

All the sections present in the analysis seem to be normal except *4s.love* and *Silvana*. If we look at the contents shown in the above image, we’ll see a bunch of text on the HEX view so we can expect some information to be there. We cannot know much more at the moment, but these 2 sections are worth to keep an eye on for later.

We can continue our static analysis (note we haven’t analyzed the running file yet) by looking into whether the file itself is packed. As stated in page 13 of [PMAL] we see that packed programs “pack” or compress the “real” or main program so that it cannot be statically analyzed. Then, an unwrapper is added on top of it so that it can be unpacked at run time and become the program it was supposed to be in memory. In other words, upon execution the unwrapper will kick in, decompress the packed code and run it as the OS itself would have done if it hadn’t been packed.

The above implies we can only really statically analyze the unpacker which isn’t of much use to us. This approach severely limits the information we can gather through a static analysis and almost always obliges us to perform a dynamic analysis of some sort.

Detecting this type of program can be done through specialized programs such as *PEiD* or by inspecting the imported functions, which can be easily done with programs such as *Dependency Walker*. A very large red flag signaling packed programs is the brevity of the imported function list. If it’s very concise we are almost certainly dealing with a packed sample as stated in page 21 of [PMAL]. After running the aforementioned tools on our sample we can conclude it’s **not packed** as it has a great deal of imported functions and it’s not deemed as packed by *PEiD* itself. We could run it through more “pack detectors” but as we are already aware of 2 factors telling us it’s **not** packed we won’t go down that rabbit hole: we need to be efficient when analyzing samples and we can’t devote time to hypothesis not backed by any evidence. We are attaching the output of both programs below.

**TODO: Attach PEiD screenshot**

**TODO: Attach Dpendency Walker screenshot**