WEEK 3 – Chapter 3: ‘Basic Dynamic Analysis’

Basic Dynamic Analysis

* Dynamic analysis is **any examination performed after executing malware**.
* Dynamic analysis techniques are the **second step** in the **malware analysis process**.
* Dynamic analysis is typically performed after **basic static analysis** has **reached** a **dead end**.
  + Could reach a dead end due to:
    - *Obfuscation*
    - *Packing*
    - *Analyst having exhausted the available static analysis techniques.*
* It can involve monitoring malware as it runs or examining the system after the malware has executed.
* Unlike static analysis, dynamic analysis **lets you observe the malware’s true functionality**, because, for example the existence of an action string in a binary does not mean the action will actually execute.
* Dynamic analysis is also an **efficient way to identify malware functionality**.
  + For example, if your malware is a **keylogger**, dynamic analysis can allow you to **locate** the **keylogger’s** **log file on the system**, **discover** the **kinds of records it keeps**, **decipher** **where** it **sends** its **information**, and so on.
  + This kind of insight would be more difficult to gain using only basic static techniques.
* Although dynamic analysis techniques are extremely **powerful**, they should be performed **only** **after basic static analysis has been completed**, because **dynamic analysis can put** your **network and system at risk**.

**Dynamic Analysis Limitation**:

* **Not all code paths may execute** when a **piece** of **malware** is **run**.
  + For example, in the case of command-line malware that requires arguments, each argument could execute different program functionality, and without knowing the options you wouldn’t be able to dynamically examine all of the program’s functionality.
* Your best bet will be to use **advanced dynamic** or **static techniques** to figure out how to force the malware to **execute all** ofits **functionality**.

Sandboxes: The Quick-and-Dirty Approach

* A ***sandbox*** is a **security mechanism** for **running untrusted programs** in a **safe environment** **without** **fear** of **harming** “*real*” **systems**.
* Sandboxes comprise virtualized environments that often simulate network services in some fashion to ensure that the software or malware being tested will function normally.
* Malware sandboxes, such as Norman Sandbox, GFI Sandbox, Anubis Joe Sandbox, Threat Expert, BitBlaze and Comodo Instant Malware Analysis will analyse malware for free. The first two are the most common.
* These sandboxes provide easy-to-understand output and are great for initial triage, as long as you are willing to submit your malware to the sandbox websites.
* Even though the sandboxes are automated, you might choose not to submit malware that contains company information to a public website.
* Beneath is the table of contents for a PDF report generated by running a file through **GFI Sandbox’s automated analysis**:



The GFI Sandbox report has **six sections**, as follows:

1. The **Analysis Summary** section
   1. lists **static** **analysis** **information** and a **high-level overview** of the **dynamic** **analysis** results.
2. The **File Activity** section
   1. lists files that are **opened**, **created**, or **deleted** for **each** **process** **impacted** by the **malware**.
3. The **Created Mutexes** section
   1. lists **mutexes** **created** **by** the **malware**.
4. The **Registry Activity** section
   1. lists **changes** **to** the **registry**.
5. The **Network Activity** section
   1. **includes** **network** **activity** **spawned** **by** the **malware**, including **setting** **up** a **listening** **port** or **performing** a **DNS** **request**.
6. The **VirusTotal Results** section
   1. lists the **results** of a **VirusTotal** **scan** of the **malware**.

Sandbox Drawbacks

* The sandbox simply **runs the executable**, **without command-line options**. If the malware executable requires command-line options, **it will not execute any code that runs only when an option is provided**.
* In addition, if your subject malware is waiting for a **command-and-control packet** to be **returned before launching a backdoor**, the **backdoor will not be launched** in the sandbox.
* The sandbox also may **not record all events**, **because neither you nor the sandbox may wait long enough**. For example, if the malware is set to **sleep** for a **day** **before** it **performs** **malicious** **activity**, you may **miss that event**. (*Most sandboxes hook the Sleep function and set it to sleep only briefly, but there is more than one way to sleep, and the sandboxes cannot account for all of these*.)
* Malware often **detects when it is running in a virtual machine**, and **if** a **virtual machine** is **detected**, the **malware might stop running** or **behave differently**.
  + *Not all sandboxes take this issue into account.*
* Some malware **requires the presence of certain registry keys or files on the system** that **might not be found in the sandbox**.
  + *These might be required to contain legitimate data, such as* ***commands*** *or* ***encryption******keys****.*
* If the **malware is a DLL**, **certain exported functions will not be invoked properly**, because a **DLL will not run as easily as an executable**.
* The **sandbox environment** OS **may not be correct for the malware**.
  + *For example, the malware might crash on Windows XP but run correctly in Windows 7.*
* A **sandbox cannot tell you what the malware does**. It may report basic functionality, but it **cannot tell you that the malware is a custom Security Accounts Manager (SAM) hash dump utility or an encrypted keylogging backdoor**.
  + *Those are conclusions that you must draw on your own.*

Running Malware

* Basic dynamic analysis techniques will be rendered useless if you can’t get the malware running.
* Here we focus on running the majority of malware you will encounter (**EXEs and DLLs**).
* Although you’ll usually find it simple enough to run executable malware by double-clicking the executable or running the file from the command line, it can be tricky to launch **malicious** **DLLs** because **Windows doesn’t know how to run them automatically**.
* The program *rundll32.exe* is **included** with **all** **modern** **versions** of **Windows**.
* It provides a container for **running** a **DLL** using **this** **syntax**:

C:\>rundll32.exe DLLname, Export arguments

* The export value must be a **function name** or **ordinal selected from the exported function table in** the **DLL**. You can use **PEview or PE Explorer to view the export table**.
* *rip.dll* contains the following exports:

Install

Uninstall

Install seems to be a likely way to launch *rip.dll*, so we can launch the malware as follows:

C:\>rundll32.exe **rip.dll**, **Install**

Malware can also have **functions** that are **exported by ordinal**—that is, as **an exported function with only an ordinal number**. In this case, you can still call those functions with *rundll32.exe* using the following command, where 5 is the ordinal number that you want to call, prepended with the # character:

C:\>rundll32.exe **xyzzy.dll**, **#5**

* Because malicious DLLs frequently run most of their code in DLLMain (*called from the DLL entry point*), and because DLLMain is executed whenever the DLL is loaded, you can often get information **dynamically by forcing the DLL to load** using *rundll32.exe*.
* Alternatively, you can even **turn a DLL into an executable** by **modifying the PE header** and **changing its extension** to **force Windows to load the DLL** as it would an **executable**.
* To modify the PE header, **wipe** the IMAGE\_FILE\_DLL *(0x2000)* flag from the **Characteristics** **field** in the IMAGE\_FILE\_HEADER.
* While this change won’t run any imported functions, it will run the DLLMain method, and it may cause the **malware** to **crash** or **terminate unexpectedly**.
* However, as long as your changes cause the **malware** to **execute its malicious payload**, and you can **collect** **information** for your **analysis**, the rest doesn’t matter.
* DLL malware may also need to be **installed as a service**, sometimes with a **convenient export such** as InstallService, as listed in *ipr32x.dll*:

C:\rundlll32 ipr32x.dll, InstallService ***ServiceName***

C:\net start ***ServiceName***

* The ServiceName argument must be provided to the malware so it can be installed and run. The net start command is used to start a service on a Windows system.

Monitoring with Process Monitor

* Process Monitor, or **procmon**, is an **advanced monitoring tool** for Windows that **provides** a **way to monitor certain registry**, file system, network, process, and thread activity. It combines and enhances the functionality of two legacy tools: FileMon and RegMon.
* Although procmon captures a lot of data, it **doesn’t capture everything**.
* For example, it can **miss the device driver activity of a user-mode component talking to a rootkit via device I/O controls**, as well as **certain GUI calls**, such as SetWindowsHookEx .
* Although procmon can be a useful tool, it usually should **not be used for logging network activity, because it does not work consistently across Microsoft Windows versions**.
* Procmon monitors all system calls it can gather as soon as it is run.
* Because many system calls exist on a Windows machine (*sometimes more than 50,000 events a minute*), it’s usually impossible to look through them all.
* As a result, because procmon uses RAM to log events until it is told to stop capturing, it can crash a virtual machine using all available memory.
* To avoid this, **run procmon for limited periods of time.**

The Procmon Display

Procmon displays **configurable columns containing information about individual events**, including the **event’s sequence number, timestamp, name of the process causing the event, event operation, path used by the event, and result of the event**.

This detailed information can be too long to fit on the screen, or it can be otherwise difficult to read. If you find either to be the case, you can view the full details of a event by double-clicking its row.



Reading the Operation column will quickly tell you which operations mm32.exe performed on this system, including registry and file system accesses. One entry of note is the creation of a file *C:\Documents and Settings\All Users\Application Data\mw2mmgr.txt* at sequence number 212 using CreateFile. The word **SUCCESS** in the Result column tells you that this operation was successful.

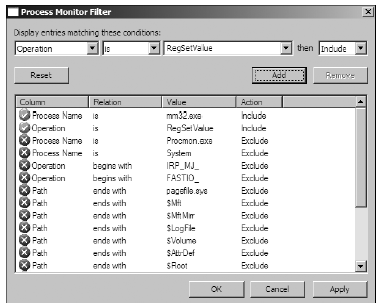
Filtering in Procmon

It’s not always easy to find information in procmon when you are looking through thousands of events, one by one. That’s where procmon’s **filtering capability** is **key**.

You can set procmon to **filter** on **one executable running on the system**. This feature is particularly useful for malware analysis, because you can set a filter on the piece of malware you are running.

You can also filter on individual system calls such as **RegSetValue**, **CreateFile**, **WriteFile**, or other suspicious or destructive calls.

To set a filter, choose **Filter** -**> Filter** to open the Filter menu.

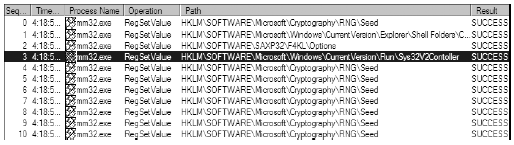


When setting a filter, first select a column to filter on using the drop-down box at the upper left, above the Reset button.

The most important filters for malware analysis are Process **Name**, **Operation**, and **Detail**.

Next, select a **comparator**, choosing from options such as **Is**, **Contains,** and **Less Than**.

Finally, choose whether this is a filter to include or exclude from display. Because, by default, the display will show all system calls, it is important to reduce the amount displayed.



As you can see in the first two rows, we’re filtering on Process **Name** and **Operation**.

We’ve added a filter on Process Name equal to *mm32.exe* that’s active when the Operation is set to RegSetValue.

After you’ve chosen a filter, click Add for each, and then click Apply. As a result of applying our filters, the display window shown in the lower image displays only 11 of the 39,351 events, making it easier for us to see that *mm32.exe* performed a RegSetValue of registry key **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Sys32V2Controller** (*sequence number 3 using RegSetValue*).

Double-clicking this RegSetValue event will reveal the data written to this location, which is the current path to the malware. If the malware extracted another executable and ran it, don’t worry, because that information is still there. Remember that the filter controls only the display.

All of the system calls that occurred when you ran the malware are captured, including system calls from malware that was extracted by the original executable. If you see any malware extracted, change the filter to display the extracted name, and then click **Apply**. The events related to the extracted malware will be displayed.

Procmon provides helpful automatic filters on its toolbar. The four filters circled filter by the following categories:



**REGISTRY** By examining registry operations, you can tell **how a piece of malware installs itself** in the **registry**.

**FILE SYSTEM** Exploring file system interaction can **show all files that the malware creates** or **configuration files it uses**.

**PROCESS ACTIVITY** Investigating process activity can tell you **whether the malware spawned additional processes**.

**NETWORK** Identifying network connections can show you **any ports on which the malware is listening**.

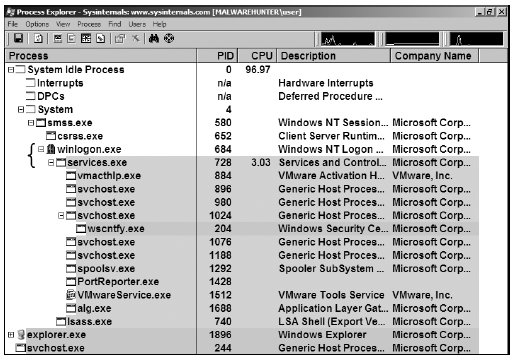
Viewing Processes with Process Explorer

The Process Explorer, **free from Microsoft**, is an extremely **powerful task manager** that should be **running** when you are **performing** **dynamic** **analysis**. It can provide **valuable insight into the processes currently running** on a **system**.

You can use Process Explorer to **list active processes**, **DLLs** loaded by a process, **various process** **properties**, and **overall system information**. You can also use it to **kill a process**, **log out users**, and **launch** and **validate processes**.

The Process Explorer Display

Process Explorer **monitors the processes running** **on** a **system** and **shows** them in a **tree structure** that **displays** **child** and **parent** **relationships**. For example, you can see that *services.exe* is a **child process** of *winlogon.exe*, as indicated by the **left curly bracket**.



***Process Explorer shows five columns:***

1. **Process** (*the process name*)
2. **PID** (*the process identifier*)
3. **CPU** (*CPU usage*)
4. **Description**
5. **Company Name**

The view updates every second. By default, services are highlighted in **pink**, processes in **blue**, new processes in **green**, and terminated processes in **red**.

Green and red highlights are **temporary** and are removed after the process has started or terminated.

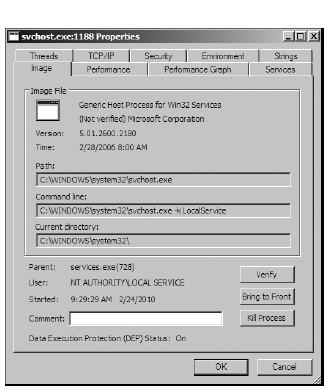
When analysing malware, watch the **Process Explorer window for changes or new processes**, and be sure to **investigate them thoroughly**.

Process Explorer can display quite a bit of information **for each process**.

For example, when the DLL information display window is active, you can **click a process to see all** **DLLs it loaded into memory**.

You can change the DLL display window to the Handles window, which shows **all handles held by the process**, including file **handles, mutexes, events**, and so on.

The Properties window shown beneath opens when you **double-click** a **process name**. This window can provide some particularly useful information about your subject malware. The Threads tab shows **all active threads**, the TCP/IP tab displays **active connections or ports on which the process is listening**, and the Image tab (*opened in the figure*) **shows the path on disk to the executable**.



Using the Verify Option

One particularly useful Process Explorer feature is the **Verify button on** the **Image tab**.

Click this button to **verify** that the image on disk is, in fact, the Microsoft signed binary. Because Microsoft uses digital signatures for most of its core executables, when Process Explorer verifies that a **signature is valid**, you can be sure that the file is actually the executable from Microsoft.

This feature is particularly useful for verifying that the Windows file on disk has **not been corrupted**; malware often replaces authentic Windows files with its own to hide. The Verify button verifies the image on disk rather than in memory, and it is useless if an attacker uses process replacement, which involves r**unning a process on the system and overwriting its memory space with a malicious executable**. Process replacement provides the malware with the same privileges as the process it is replacing, so that the malware appears to be executing as a legitimate process, but it leaves a fingerprint: The image in memory will differ from the image on disk. For example, above the *svchost.exe* process is verified, yet it is actually **malware**.

Using Dependency Walker

Process Explorer allows you to launch *depends.exe (Dependency Walker)* on a **running process** by right-clicking a process name and selecting **Launch Depends**.

It also lets you search for a **handle** or **DLL** by choosing **Find -> Find Handle** or **DLL**.

The Find DLL option is particularly useful when you find a malicious DLL on disk and **want to know** if **any running processes use that DLL**.

The Verify button **verifies the EXE** file on disk, but **not every DLL loaded during runtime**. To determine whether a DLL is loaded into a process after load time, you can **compare the DLL list in Process Explorer** to the **imports shown in Dependency Walker**.

Analysing Malicious Documents

You can also use Process Explorer to **analyse malicious documents**, such as **PDFs** and **Word** **documents**.

A quick way to determine whether a document is malicious is to **open Process Explorer and then open the suspected malicious document**. If the document **launches** any **processes**, you should see them in Process Explorer, and be able to **locate the malware on disk via the Image tab** of the **Properties window**.

Comparing Registry Snapshots with Regshot

Regshot is an **open source registry comparison tool** that **allows you to take and compare two registry snapshots**.

To use Regshot for malware analysis, simply take the first shot by clicking the **1st Shot** button, and **then run the malware and wait for it to finish making any system changes**.

Next, take the second shot by clicking the **2nd Shot** button. Finally, **click** the **Compare button to compare the two snapshots**.

The listing beneath displays **a subset of the results** **generated by Regshot** during **malware analysis**.

Registry snapshots were taken **before** and **after** **running** the **spyware *ckr.exe***.

Regshot Comparison:

Regshot

Comments:

Datetime: <date>

Computer: MALWAREANALYSIS

Username: username

----------------------------------

Keys added: 0

----------------------------------

----------------------------------

Values added:3

----------------------------------

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ckr:C:\WINDOWS\system32\

ckr.exe

...

...

----------------------------------

Values modified: 2

----------------------------------

HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed: 00 43 7C 25 9C 68 DE 59 C6 C8

9D C3 1D E6 DC 87 1C 3A C4 E4 D9 0A B1 BA C1 FB 80 EB 83 25 74 C4 C5 E2 2F CE

4E E8 AC C8 49 E8 E8 10 3F 13 F6 A1 72 92 28 8A 01 3A 16 52 86 36 12 3C C7 EB

5F 99 19 1D 80 8C 8E BD 58 3A DB 18 06 3D 14 8F 22 A4

...

----------------------------------

Total changes: 5

----------------------------------

As you can see *ckr.exe* creates a **value** at:

* *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* as a **persistence mechanism**.

A certain amount of noise is typical in these results, because the **random-number generator seed is constantly updated in** the **registry**.

As with **procmon**, your **analysis** of these results **requires patient scanning** **to find nuggets of interest**.