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 particular 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.