

# Introduction à la virologie

borello.jeanmarie@gmail.com

# Introduction

- **Malware**, contraction de **Malicious** et de **Software** = code malveillant
- **Malveillant**
  - notion non formalisable : suite d'appels « légitime » de l'OS
  - Suite d'actions « illégitime » pour l'utilisateur et les outils de détection
  - ➔ Particularité au niveau du code
    - Protection du code
    - Furtivité
    - Persistance
- Comment les détecter
- panorama

# Plan

- **Spécificités techniques des malwares**
  - Primo-infection
  - Persistance
  - Furtivité
  - Protection contre l'analyse
- **Détection des malwares**
  - Détection statique
  - Détection dynamique
- **Panorama des malwares**
  - Chevaux de Troie
  - Vers et virus
  - Rootkit/bootkit
  - Botnets
  - ransomware

# Spécificités techniques des malwares

Primo-infection

Persistence

Furtivité

Protection contre l'analyse

# Primo-infection

- **Social engineering**
  - **Phishing** (hameçonnage) : campagne de mail massive pour inciter les utilisateurs à ouvrir une pièce jointe malveillante ou à fournir des données personnelles
  - **Spear phising** (harponnage) : phishing ciblé
- **Water holing** (attaque du point d'eau) : compromission d'un site utilisé par la/les victime(s)
- **Vulnérabilités** : exploitation d'un service vulnérable afin de réaliser une exécution de code distante et ou privilégiée

# Persistence

- Consiste pour le malware à assurer sa survie sur le système en cas de redémarrage
- Multiples techniques de base
  - Registre
  - Tâches planifiées
  - Services,
  - Shell
  - ...
- Point de départ: **autoruns.exe** de la suite SysInternals

# Persistence

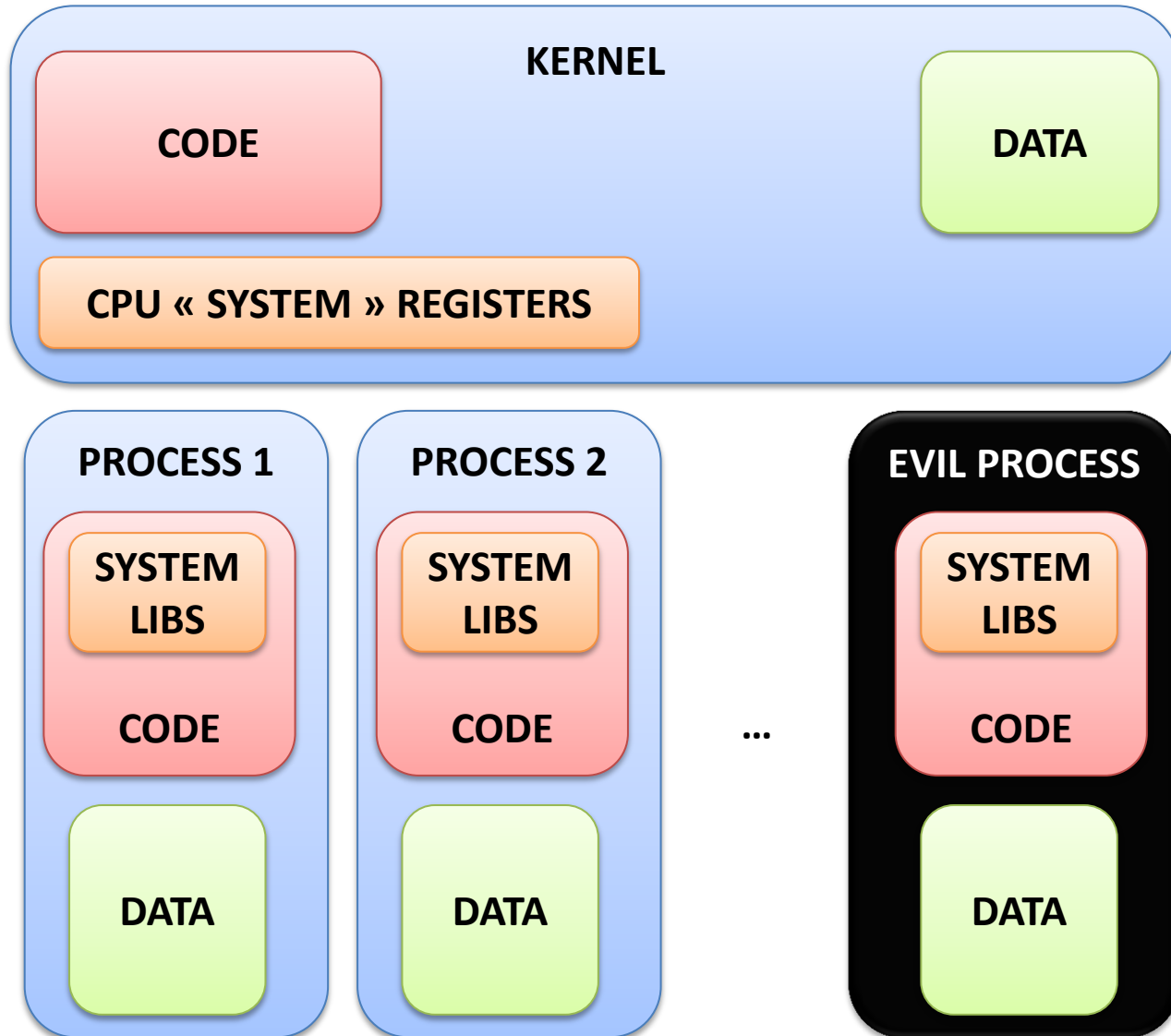
- Techniques plus avancées:
  - **Hijacking** : remplacer un élément légitime par un autre illégitime
    - DLL/EXE
    - COM
    - raccourcis
  - **Association de fichiers** : via plusieurs clés de registre
  - **Squatting** : ajouter un élément malveillant recherché par défaut par un logiciel légitime voir le système lui-même
  - **DLL search order**:
    1. Le répertoire où l'application est installée
    2. Le répertoire système (%SystemRoot%\System32)
    3. Le répertoire de windows (%SystemRoot%)
    4. Le répertoire courant
    5. Les répertoires définis dans la variable d'environnement PATH

# Furtivité

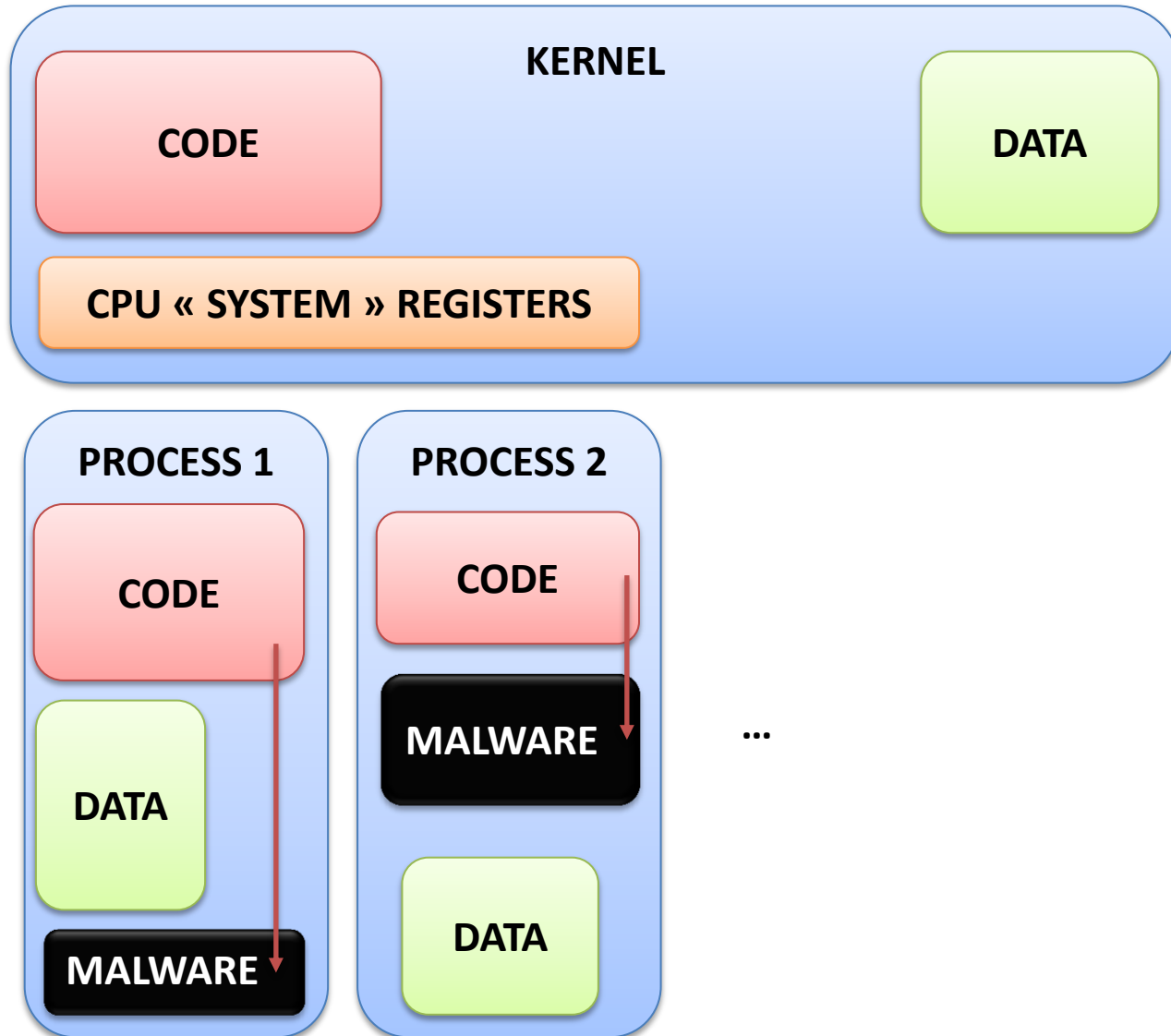
- Consiste à passer inaperçu pour l'utilisateur et les outils de détection : Non détection
- Techniques
  - **Mimétisme** : trojan
  - **Infection** : virus
  - **Modifications** de l'OS : rootkits
  - **Minimisation des interactions** avec l'OS



# Furtivité : mimétisme

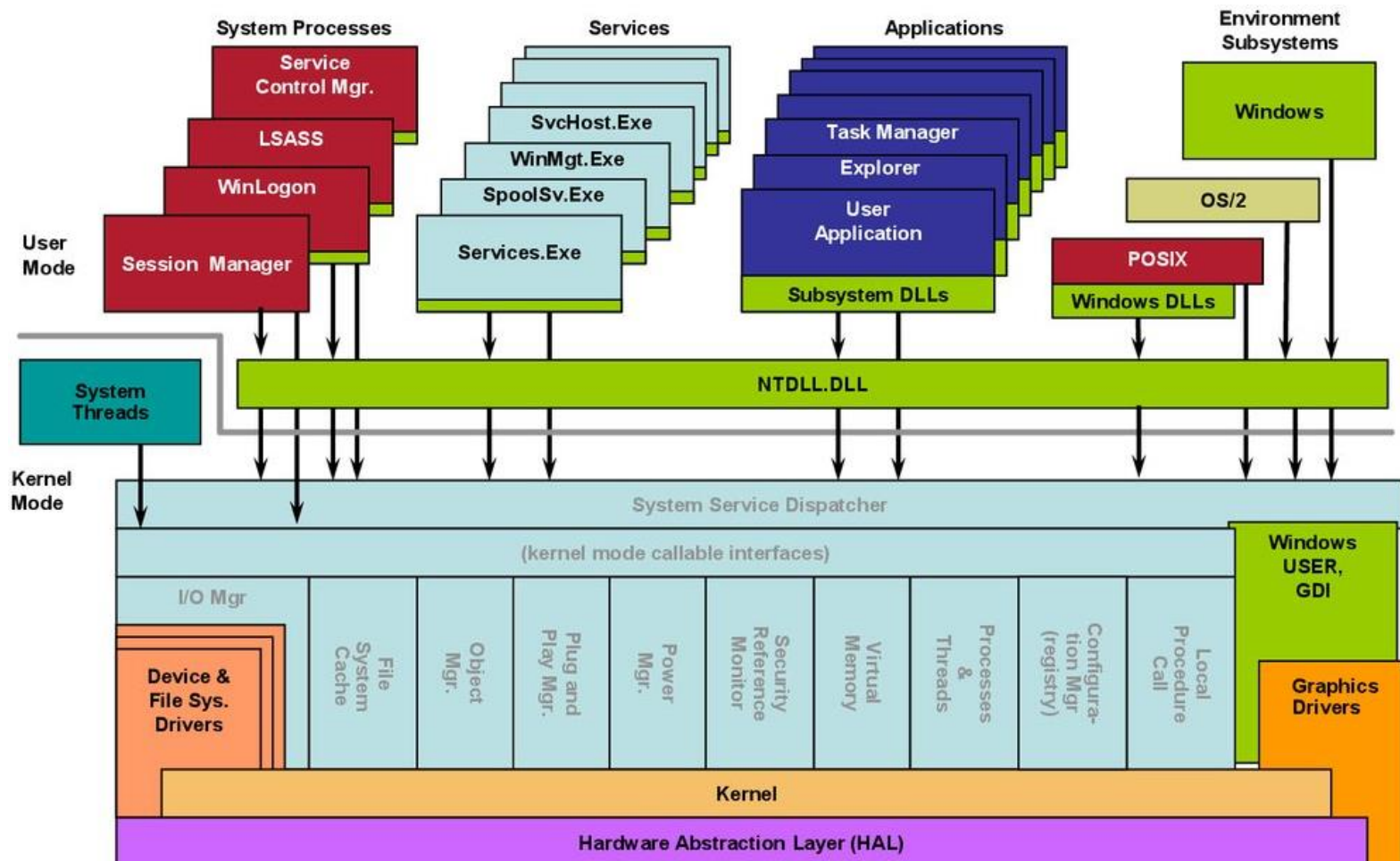


# Furtivité : infection



# Furtivité : interactions

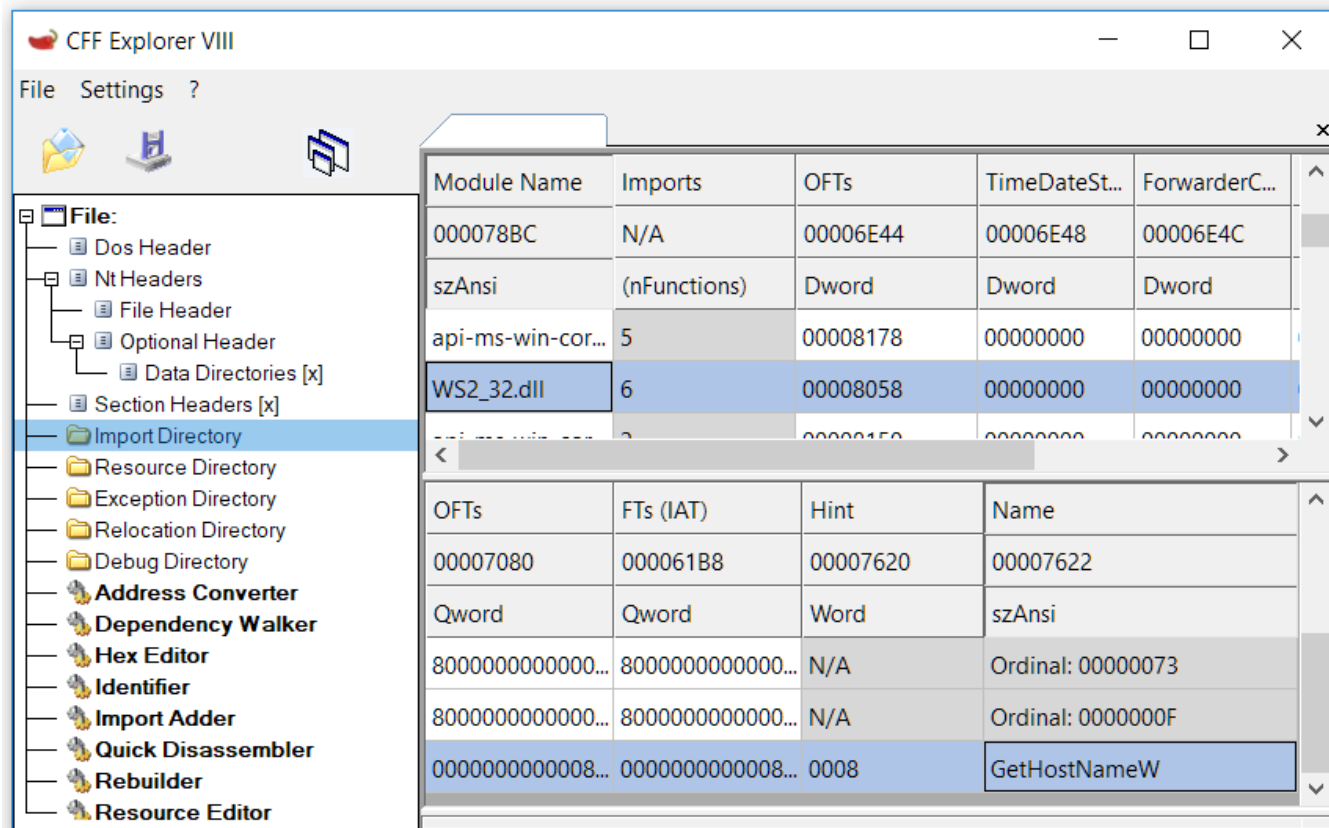
- Minimisation des interactions avec l'OS



# Furtivité : interactions

– Exemple : **GetHostNameW** dans **ws2\_32.dll**

## 1. Appel direct



CFF Explorer VIII

File Settings ?

File:

- Dos Header
- Nt Headers
  - File Header
  - Optional Header
    - Data Directories [x]
  - Section Headers [x]
- Import Directory
- Resource Directory
- Exception Directory
- Relocation Directory
- Debug Directory
- Address Converter
- Dependency Walker
- Hex Editor
- Identifier
- Import Adder
- Quick Disassembler
- Rebuilder
- Resource Editor

Module Name	Imports	OFTs	TimeDateSt...	ForwarderC...
000078BC	N/A	00006E44	00006E48	00006E4C
szAnsi	(nFunctions)	Dword	Dword	Dword
api-ms-win-cor...	5	00008178	00000000	00000000
WS2_32.dll	6	00008058	00000000	00000000

OFTs	FTs (IAT)	Hint	Name
00007080	000061B8	00007620	00007622
Qword	Qword	Word	szAnsi
80000000000000...	80000000000000...	N/A	Ordinal: 00000073
80000000000000...	80000000000000...	N/A	Ordinal: 0000000F
00000000000000...	00000000000000...	0008	GetHostNameW

# Furtivité : interactions

## 2. Résolution dynamique via **GetModuleHandleA** et **GetProcAddress**

```
//...  
HMODULE hWinsock = GetModuleHandleA("ws2_32.dll");  
FARPROC GetHostNameFn = GetProcAddress(hWinsock, "GetHostNameW");  
//...
```

# Furtivité : interactions

## 2. GetProcAddress recodé (on peut faire de même pour GetModuleHandleA)

```
FARPROC WINAPI SelfGetProcAddress(HMODULE hModule, ULONG ProcHashValue){
    // declarations omitted...
    pImageDosHeader = (IMAGE_DOS_HEADER*)hModule;
    pImageNtHeaders = (IMAGE_NT_HEADERS*)(pModuleBase+pImageDosHeader->e_lfanew);
    pImageExportDirectory =
        (pBase+pImageNtHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

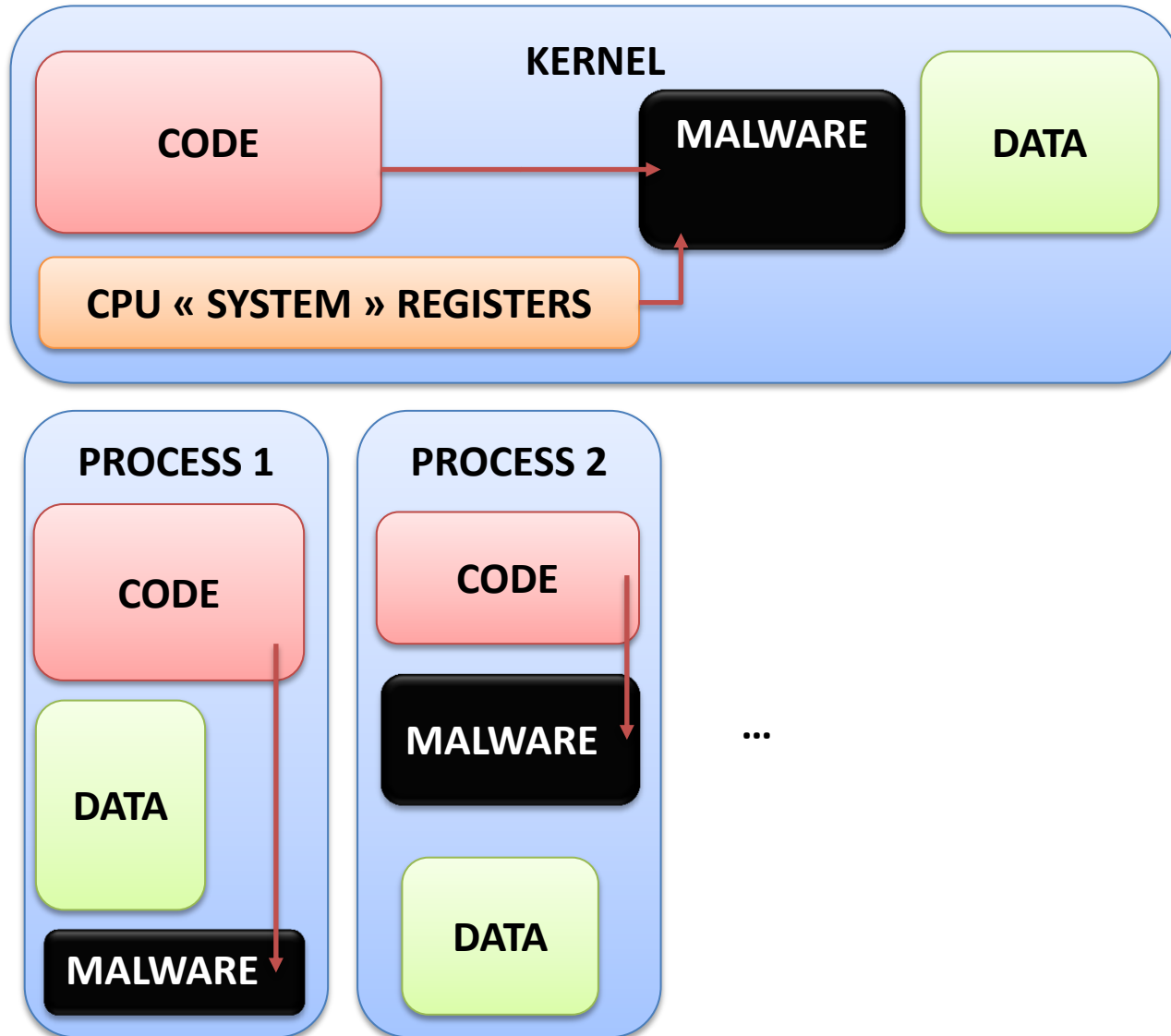
    AddressOfFunctionsRvaTbl= (DWORD*) (pBase+pImageExportDirectory->AddressOfFunctions);
    AddressOfNamesRvaTbl    = (DWORD*) (pBase+pImageExportDirectory->AddressOfNames);
    AddressOfNameOrdinalsTbl= (WORD*)  (pBase+pImageExportDirectory->AddressOfNameOrdinals);

    for(ExportIndex=0; ExportIndex < pImageExportDirectory->NumberOfNames; ExportIndex++){
        FunctionName = pBase+AddressOfNamesRvaTbl[ExportIndex];
        if (crc32((const char*)FunctionName) == ProcHashValue){
            OrdinalIndex = AddressOfNameOrdinalsTbl[ExportIndex];
            ProcAddress  = (FARPROC)(pBase + AddressOfFunctionsRvaTbl[OrdinalIndex]);
            goto End;
        }
    }
    End:
    return (ProcAddress);
}
```

# Furtivité : modification de l'OS

- Différents niveaux de furtivité (*Joanna Rutkowska*)
  - Type1 : modification des constantes du système
  - Type2 : modification des variables du système
  - Type3 : virtualisation du système

# type 1 : modification des constantes

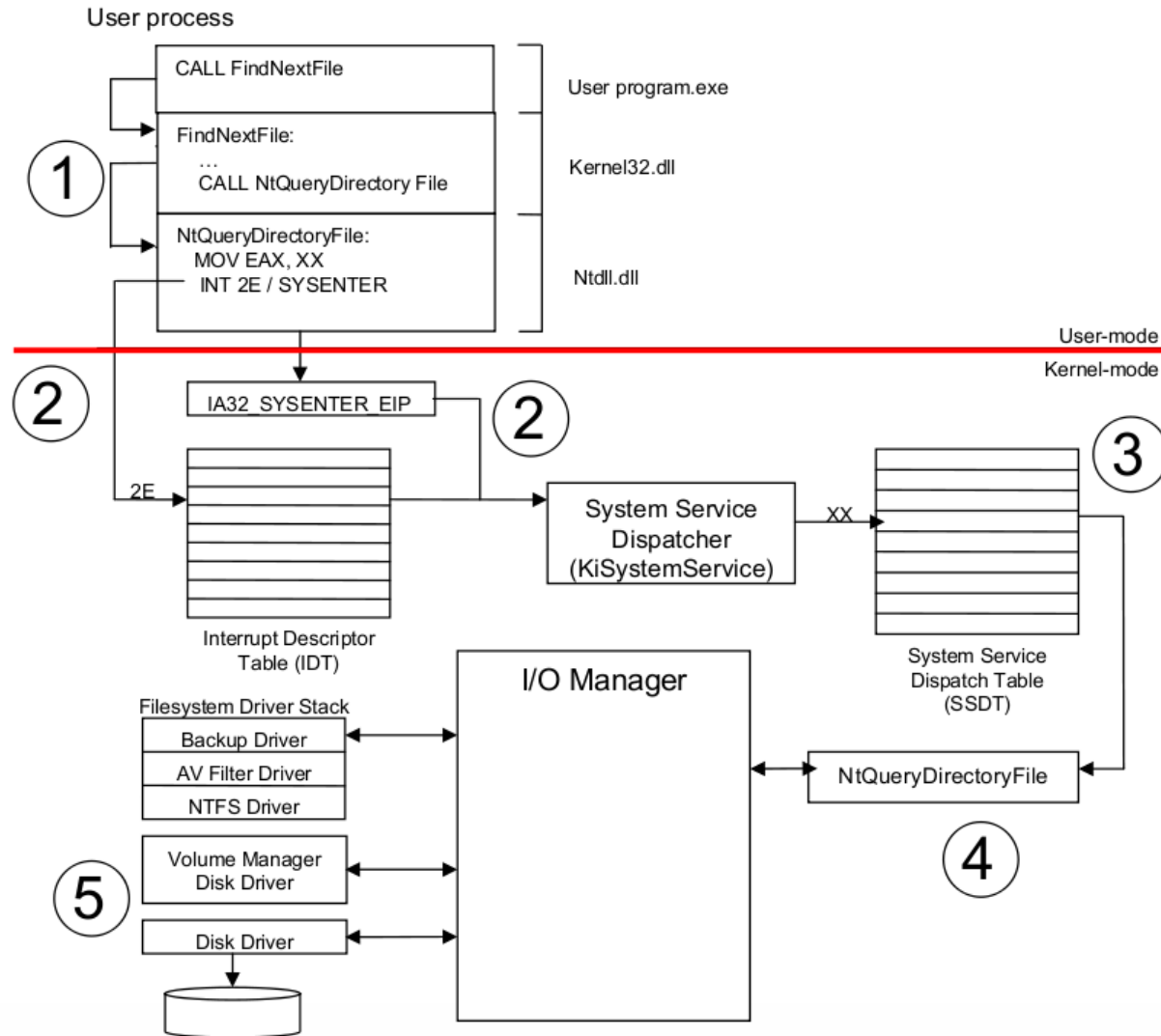




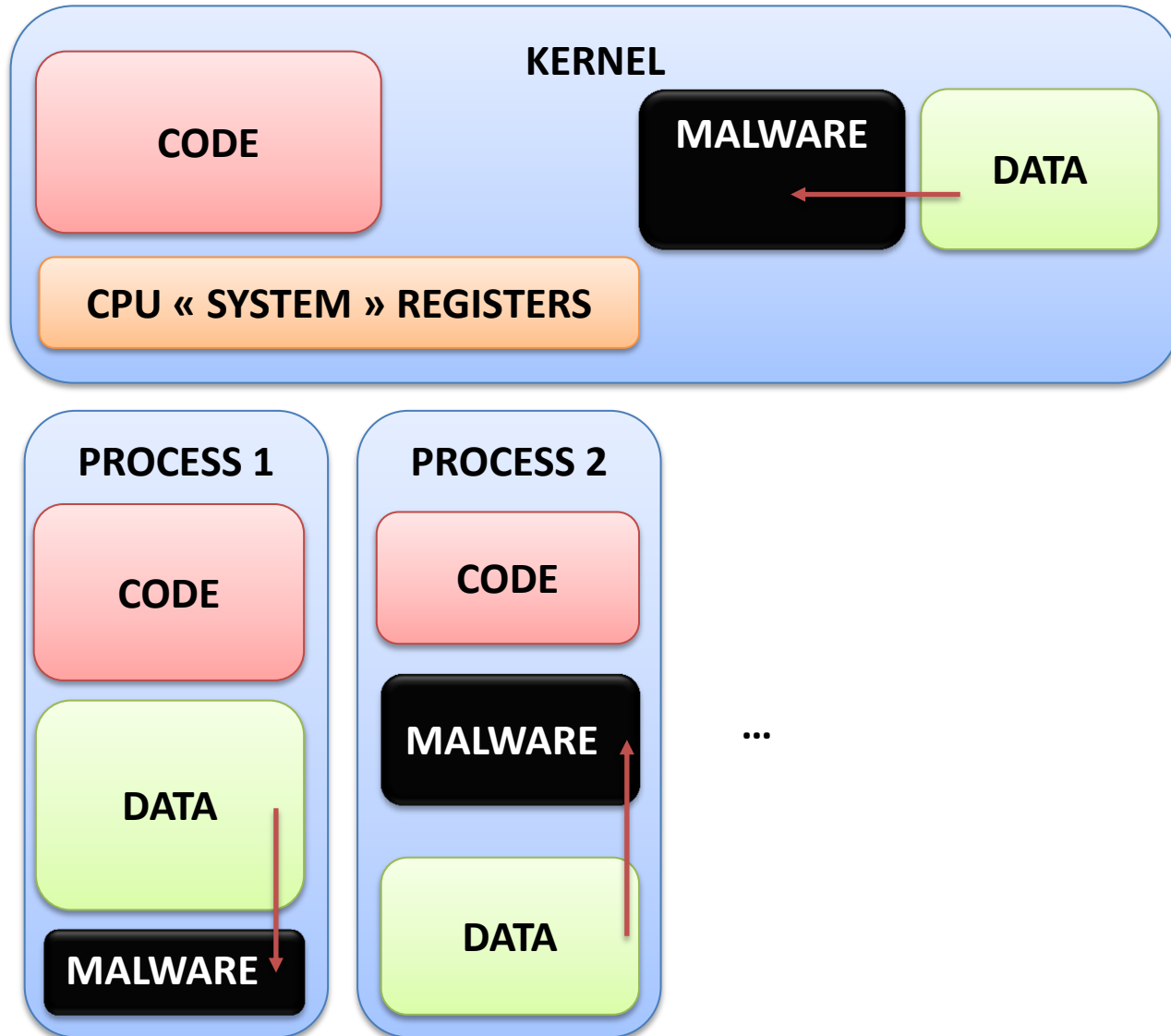
# type 1 : modification des constantes

- Fonctionnement:
  - Kernel Land: hook des principales tables du noyaux
    - SSDT
    - IDT
    - Drivers
  - User Land :
    - hook de l'IAT
    - Injection de DLL

# type 1 : modification des constantes



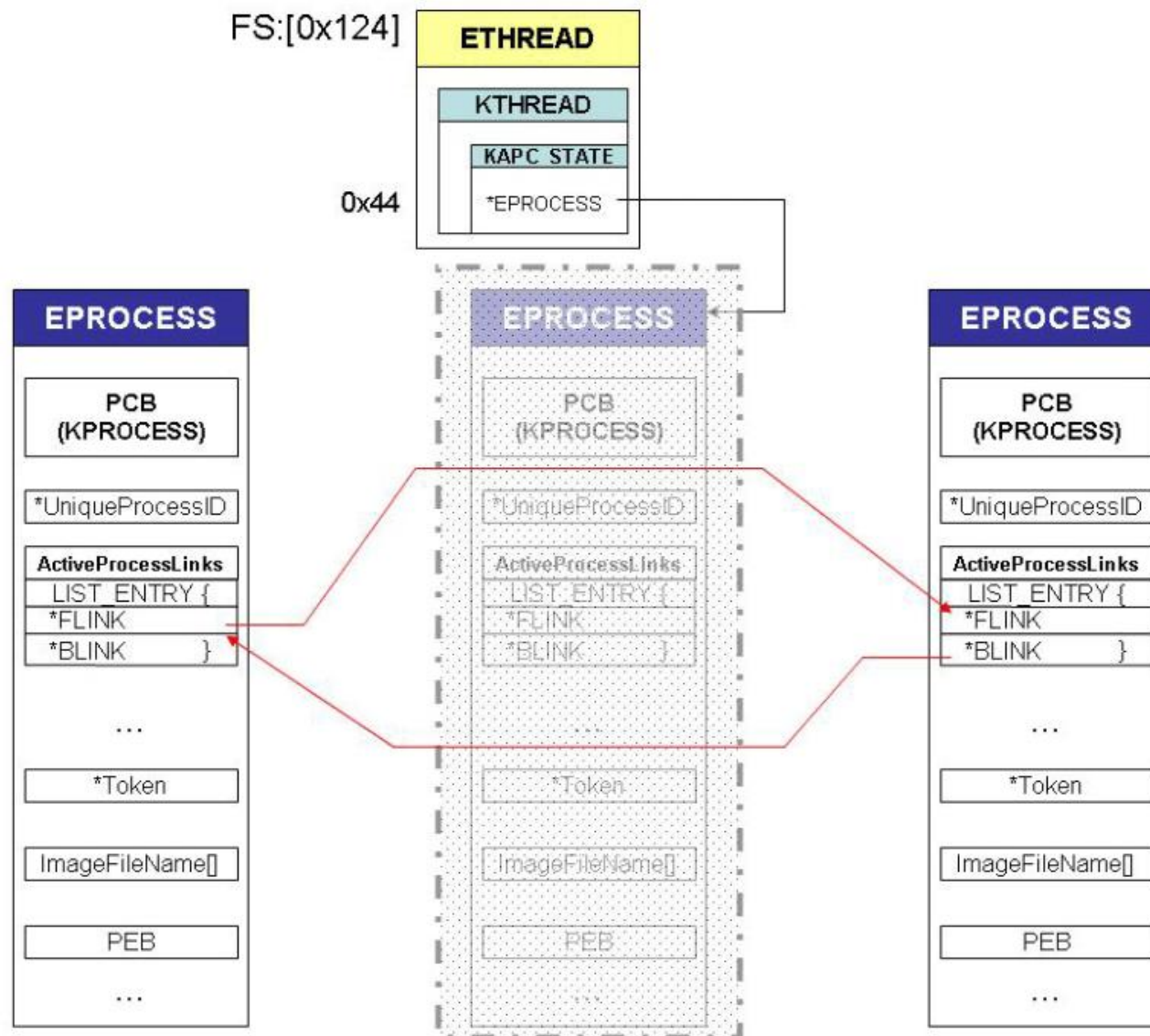
# type 2 : modification des données



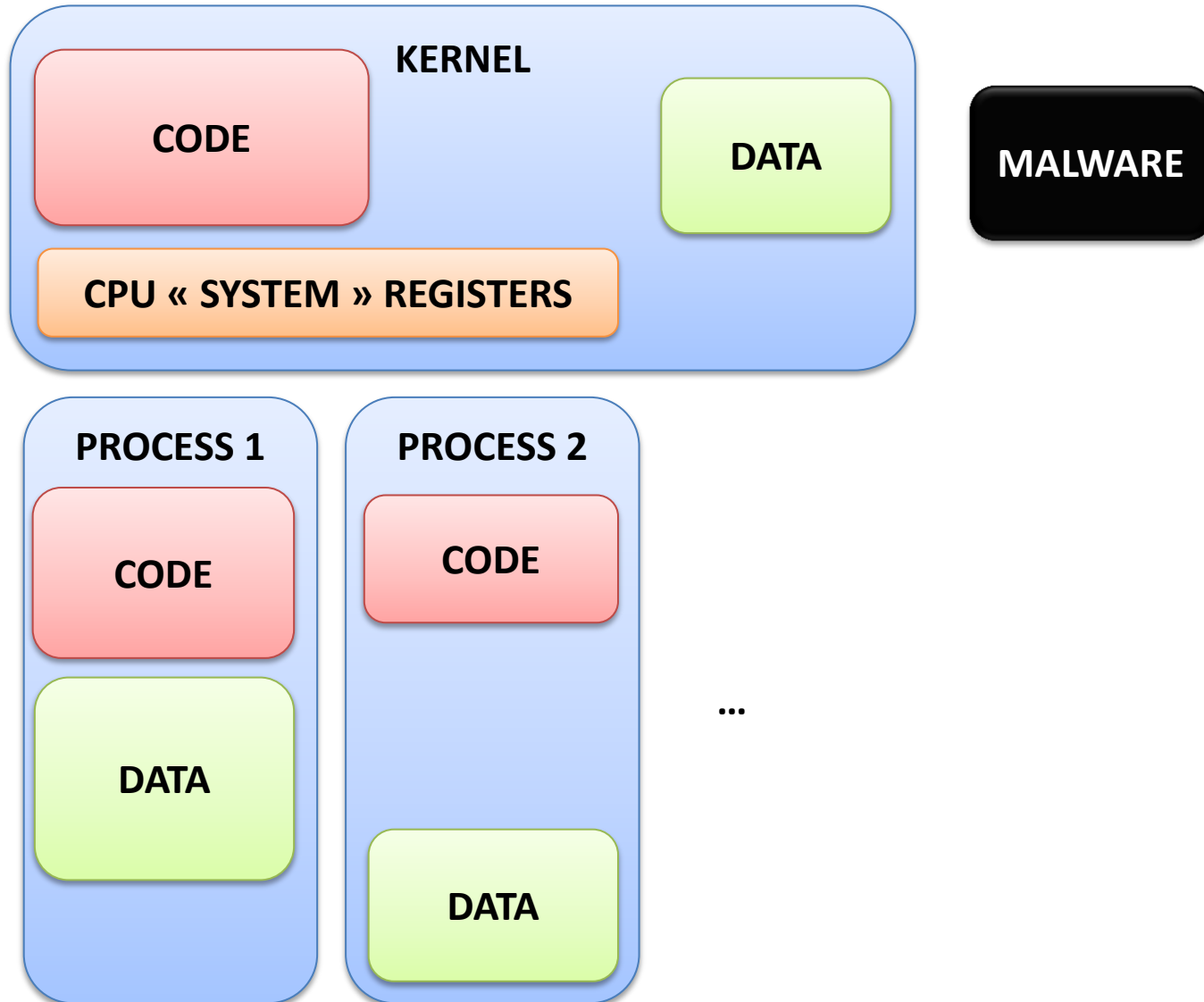
# type 2 : modification des données

- Fonctionnement:
  - Exclusivement Kernel Land
  - DKOM (*Direct Kernel Object Modification*)
- Exemples:
  - Modification de la liste chaînée des processus (EPROCESS)

# type 2 : modification des données



# type 3 : virtualisation du système



# type 3 : virtualisation du système

- Fonctionnement:
  - Utilisation des instructions de virtualisation pour virtualiser l'OS (AMD-V et Intel VT-x)



**Le rootkit ne peut pas utiliser de service de l'OS  
→ il doit interagir directement avec le matériel**

- Exemples:
  - SubVirt → lancement de l'OS dans une machine virtuelle
  - BluePill (AMD-V) et Vitriol (Intel VT-x)

# Protection contre l'analyse

- Consiste à se prémunir autant que possible d'une analyse (statique et dynamique)
  - Protection du code
    - Chiffrement
    - Polymorphisme
    - Métamorphisme
    - Packers
  - Détection des débogueurs
  - Détection d'instrumentation de code (DBI)
  - Détection de VM



# Chiffrement

- Technique ancienne (1990) : Cascade.1701
- Idée : déchiffrer le corps du virus, l'exécuter et le rechiffrer avec une nouvelle clé.
- Chiffrement faible mais suffisant pour contourner une détection par signature

```
                lea si, Start
                mov sp, 0682
Decrypt:        xor [si],si
                xor [si],sp
                inc si
                dec sp
                jnz Decrypt
```

**Start:**

*; Encrypted/Decrypted Virus Body*

- Mais la routine de déchiffrement est un motif potentiel de détection

# Polymorphisme

1/4

- Technique apparue en 1990 : virus 1260
- Idée : en plus du chiffrement, changer la forme de la routine de déchiffrement
- **Insertion de code mort**
- Substitution de variables
- Permutation d'instructions
- Substitution d'instructions

Exemple	Signification
add reg, 0	$\text{reg} \leftarrow \text{reg} + 0$
mov reg, reg	$\text{reg} \leftarrow \text{reg}$
or reg, 0	$\text{reg} \leftarrow \text{reg} \mid 0$
and reg, -1	$\text{reg} \leftarrow \text{reg} \& -1$

# Polymorphisme

2/4

- Technique apparue en 1990 : virus 1260
- Idée : en plus du chiffrement, changer la forme de la routine de déchiffrement
- Insertion de code mort
- **Substitution de variables**
- Permutation d'instructions
- Substitution d'instructions

Programme1	Programme2
pop <b>edx</b>	pop <b>eax</b>
mov <b>edi</b> , 04h	mov <b>ebx</b> , 04h
mov <b>esi</b> , ebp	mov <b>edx</b> , ebp
mov <b>eax</b> , 0Ch	mov <b>edi</b> , 0Ch
add <b>edx</b> , 088h	add <b>eax</b> , 088h

# Polymorphisme

3/4

- Technique apparue en 1990 : virus 1260
- Idée : en plus du chiffrement, changer la forme de la routine de déchiffrement
- Insertion de code mort
- Substitution de variables
- **Permutation d'instructions**
- Substitution d'instructions

Programme1	Programme2
mov ecx, 104h	mov edi, [rbp+08h]
mov esi, [ebp+0xh]	mov ecx, 104h
mov edi, [rbp+08h]	mov esi, [ebp+0xh]
repnz movsb	repnz movsb

# Polymorphisme

4/4

- Technique apparue en 1990 : virus 1260
- Idée : en plus du chiffrement, changer la forme de la routine de déchiffrement
- Insertion de code mort
- Substitution de variables
- Permutation d'instructions
- **Substitution d'instructions**

Instruction simple	Instructions multiples
xor reg, reg	mov reg, 0
mov reg, Imm	push imm pop reg
op reg1, reg2	mov mem, reg op mem, reg2 mov reg, mem

# Métamorphisme

1/2

- Idée : muter l'intégralité du code
- Ex: Win32.MetaPHOR, réplication en 5 étapes
  1. Désassemblage et dépermutation : représentation de ses instructions en pseudo-code
  2. Compression : utilisation de règles de réécritures (multiples instr. → simple instr.)
  3. Permutation du code au moyen de saut inconditionnels
  4. Extension : utilisation de règles de réécritures (simple instr. → multiples instr.)
  5. Assemblage : production du nouveau binaire à partir du pseudo-code.

# Métamorphisme

2/2

Binaire1

Désassemblage  
linéaire

Code assembleur 1

```
FF 35 64 B0
44 00 8F 05
74 B8 44 00
89 05 74 B8
44 00 FF 35
74 B8 44 00
8F 05 64 B0
44 00 8D 0D
04 00 00 00
FF F1 BD 00
10 00 00 89
2D A4 B8 44
00 FF 35 A4
B8 44 00 C7
05 F6 B1 44
00 00 C0 34
00 8B 3D F6
B1 44 00 57
6A 00 8F C6
81 CF 00 00
00 00 FF F6
C7 C1 64 B0
44 00 FF 31
8F 05 AD B4
44 00 FF 15
AD B4 44 ...
```

```
push dword_44B064
pop dword_44B874
mov dword_44B874, eax
push dword_44B874
pop dword_44B064
lea ecx, large ds:4
push ecx
mov ebp, 1000h
mov dword_44B8A4, ebp
push dword_44B8A4
mov dword_44B1F6, 34C000h
mov edi, dword_44B1F6
push edi
push 0
pop esi
or edi, 0
push esi
mov ecx, offset dword_44B064
push dword ptr [ecx]
pop dword_44B4AD
call dword_44B4AD
```

Compression

archétype

```
push 4
push 1000h
push 34C000h
push 0
call VirtualAlloc
```

Extension

Code assembleur 2

```
push eax
pop dword_1007088
mov dword_100729A, 0
xor ebp, dword_100729A
push 4
pop ebp
push ebp
mov ebx, 1000h
mov dword_100721B, ebx
push dword_100721B
push 340000h
pop esi
push esi
mov ebx, 655574FFh
xor ebx, 655574FFh
push ebx
lea esi, dword_1007088
call dword ptr [esi+0]
```

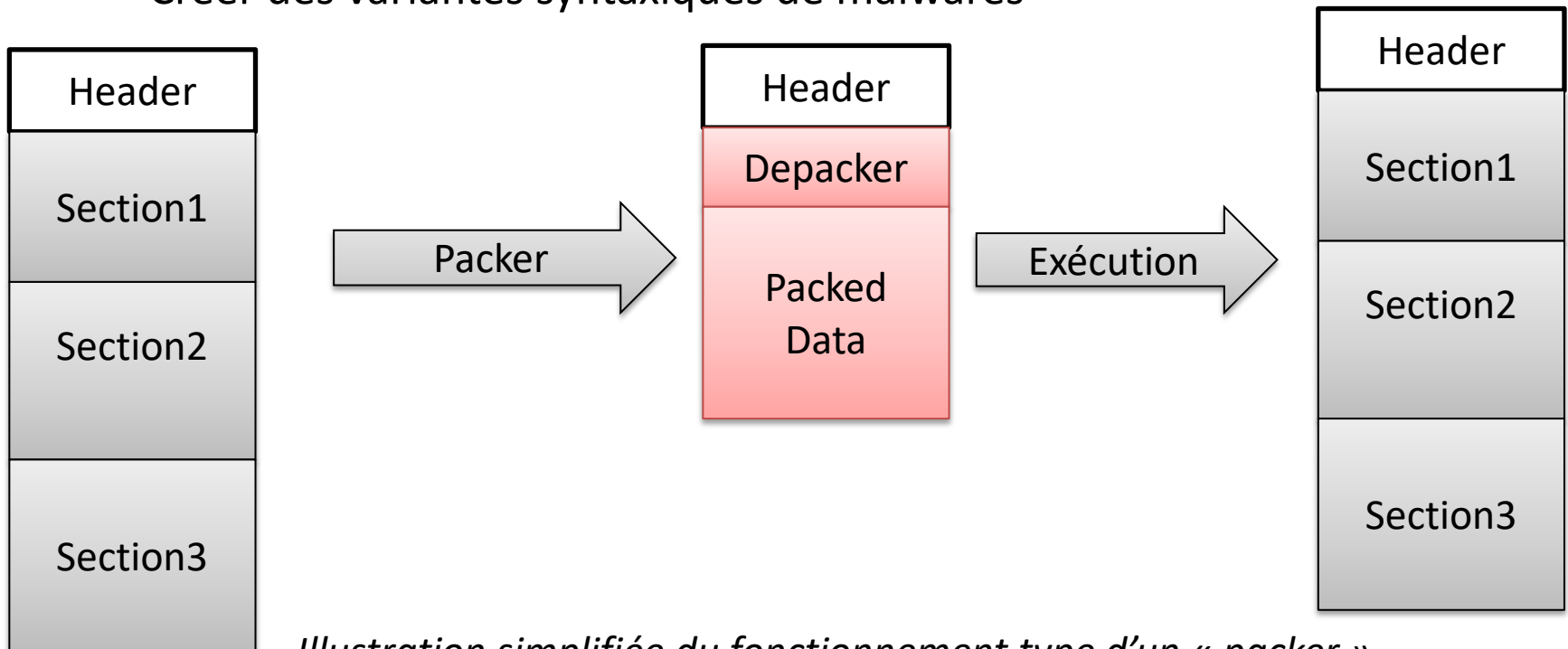
Assemblage et  
attachement

Binaire2

```
50 8F 05 88
70 00 01 C7
05 9A 72 00
01 00 00 00
00 33 2D 9A
72 00 01 6A
04 8F C5 FF
F5 BB 00 10
00 00 89 1D
1B 72 00 01
FF 35 1B 72
00 01 68 00
00 34 00 8F
C6 56 C7 C3
FF 74 55 65
81 F3 FF 74
55 65 53 8D
35 88 70 00
01 FF 54 26
00
```

# « Packers »

- À l'origine conçus pour diminuer la taille d'un programme
- Aujourd'hui utilisés pour :
  - Protéger la propriété intellectuelle (ralentir la rétro-conception)
  - Créer des variantes syntaxiques de malwares



*Illustration simplifiée du fonctionnement type d'un « packer »*

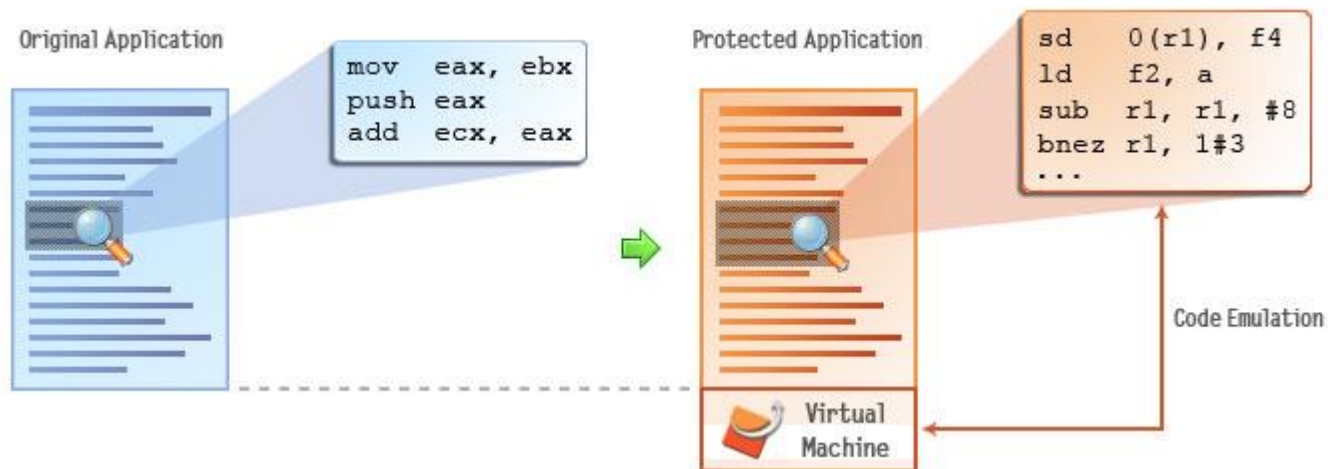


# « Packers »

- Exemple simple : UPX
  - Suppression des imports d'origines
    - Kernel32.dll
      - LoadLibraryA
      - GetProcAddress,
      - VirtualAlloc/Free/Protect
      - ExitProcess
  - Suppression des sections d'origine
    - UPX0,
    - UPX1
    - UPX2
    - .idata pour les imports

# « Packers »

- Exemple complexe : VMProtect



— Fonctionne au moyen de marqueurs

```
#include "VMProtectSDK.h"
```

```
VMProtectBegin(MARKER_NAME);
```

```
...
```

```
VMProtectEnd();
```

# Détection des débogueurs

- Détection de points d'arrêt
- API
- Flags de debug
- Timing
- Auto-debug

# Détection des débogueurs

- Détection de points d'arrêt:
  - **Logiciels** : 0xCC pour x86
    - Vérification de la présence de 0xCC à la place du code d'origine
    - Plus globalement, vérification de l'intégrité du code (si non auto-modifiant)
  - **Matériels** : registres DR
    - Récupération et modification des registres de debug via GetThreadContext et SetThreadContext
    - Même chose via la gestion d'exception (SEH) qui permet d'accéder au contexte du thread

# Détection des débogueurs

- API de détection
  - **IsDebuggerPresent** : détermine si le processus appelant est en cours de debug
  - **CheckRemoteDebuggerPresent** : détermine si le processus spécifié est en cours de debug
  - **NtCreateDebugObject/NtQueryObject** : création d'un objet de debug pour un processus donné, ensuite vérification du nombre de handle sur cet objet : 1 pas de debug, >1 debug
  - **CloseHandle/NtClose** : génère une exception STATUS\_INVALID\_HANDLE sur un handle invalide en cas de debug

# Détection des debuggers

- Flags de debug:
  - **TrapFlag** : TF du registre EFLAGS, lorsque ce flag est à 1, le CPU génère une interruption 1 après l'exécution d'une instruction (single step)
  - **BeingDebugged** : flag dans le PEB
  - **KdDebuggerEnabled**: flag de la structure KUSER\_SHARED\_DATA, permet de savoir si le debug kernel est actif

# Détection des debuggers

- Timing
  - **GetTickCount** : nombre de millisecondes depuis de dernier démarrage du CPU
  - **QueryPerformanceCounter** : compteur « haute performance » (<1us)
  - **rdtsc (Read Time Stamp Counter)** : instruction permettant de connaître le nombre de *ticks* du CPU depuis le dernier démarrage

# Détection des debuggers

- « auto-debug »
  - Processus fils débogué par un processus père
  - ➔ **DebugActiveProcess()** échoue sur le fils



# Détection de DBI

- **Dynamic Binary Instrumentation (DBI):**  
framework d'injection de code pour analyser un programme
  - **PIN**, framework propriétaire développé par Intel pour IA-32 et x86-64
  - **DynamoRIO**, framework open-source pour x86/AMD64/ARM/AArch64



# Détection de DBI

- **eXait** de coresecurity permet de détecter PIN de plus de 15 façons différentes
  - Chaînes de caractères
  - Présence du module pinvm.dll
  - Noms d'évènements et de sections (PIN\_IPC...)
  - Détection du compilateur JIT
    - ntdll.dll hooks
    - Page permission RWX

# Détection de VM

Par défaut les machines virtuelles ne cherchent pas à être furtives à travers:

- Les instructions émulées
- Les processus et fichiers dédiés
- Leurs configurations
- Leur matériel

# Détection de VM

Instructions:

- **CPUID**, EAX=1 : propriétés du processeur dans ECX avec bit 31 à 0 pour une machine physique et 1 pour une VM
- **CPUID**, EAX = 0x40000000 : « hypervisor brand » (Microsoft HV, VMwareVMware, etc)
- **IN** avec EAX='VMXh' et EDX='VX', fonctionne uniquement avec VMWare, exception sinon

# Détection de VM

- Processus:
  - Vmwareuser.exe
  - VBoxservice.exe
  - ...
- Drivers:
  - Vmmouse.sys
  - VBoxMouse.sys
  - VBoxGuest.sys
  - ...

# Détection de VM

- Registre
  - HKEY\_LOCAL\_MACHINE\HARDWARE\ACPI\DSDT\VBOX\_\_
  - HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\VirtualDeviceDrivers
  - HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Enum\SCSI\
  - ...

# Détection de VM

- Matériel: préfixe des adresses MAC
  - 00:05:69 (Vmware)
  - 00:0C:29 (Vmware)
  - 00:1C:14 (Vmware)
  - 00:50:56 (Vmware)
  - 08:00:27 (VirtualBox)

# Détection des malwares

Statique

Dynamique



# Détection : définitions

- $P$  ensemble des programmes et  $M \subset P$  ensemble des malwares
- Fonction de détection  $D: P \rightarrow \{0,1\}$  définie par
  - $\forall p \in P, \begin{cases} D(p) = 1, & \text{si } p \in M & (\text{un positif}) \\ D(p) = 0 & \text{sinon} & (\text{un négatif}) \end{cases}$

Détecteur parfait

		Positif	Négatif
Détecteur réel	Positif	Vrai Positif (VP)	Faux Positif (FP)
	Négatif	Faux Négatif (FN)	Vrai Négatif (VN)
		$P$	$N$

**Fiabilité** =  $\frac{VP}{P}$   
(tout ce qui doit être détecté l'est)

**Pertinence** =  $\frac{VN}{N}$   
(seul ce qui doit être détecté l'est)

# Techniques de détection

- 2 grandes approches de détection

Statique		Dynamique
Avantages	<ul style="list-style-type: none"><li>• Pas de risque liés à l'exécution</li><li>• Rapide</li><li>• Efficace sur des programmes simples</li><li>• Facile à déployer</li></ul>	<ul style="list-style-type: none"><li>• Résiste aux mutations de code</li><li>• Peut détecter de nouvelles menaces</li></ul>
Inconvénients	<ul style="list-style-type: none"><li>• Peu robuste face aux techniques de mutation de code</li><li>• Nécessite une base de connaissances (signatures) importantes</li><li>• Peu robuste face à de nouvelles menaces</li></ul>	<ul style="list-style-type: none"><li>• Exécution liée à l'environnement</li><li>• Analyse un unique chemin</li><li>• Difficile à mettre en place</li><li>• Potentiellement risqué (exécution du code malveillant)</li></ul>

# Détection statique

- Signatures
  - Motif constant (expression régulière) permettant d'identifier un code binaire
    - Si le motif est présent dans un autre binaire → faux positif
    - Si le motif n'est pas trouvé dans une variante → faux négatif
- Inadaptées pour les codes mutants (auto-modifiants ou polymorphes)



# Détection statique

- Heuristiques
  - Point d'entrée dans la dernière section
  - Attributs de section suspect (W+X)
  - Taille virtuel incorrect
  - Redirection du point d'entrée (JMP)
  - Nom de section (exécution en .reloc ou .debug, etc)
  - Imports par ordinaux de la Kernel32.dll
  - Inconsistance dans les headers (taille du code / données)
  - etc

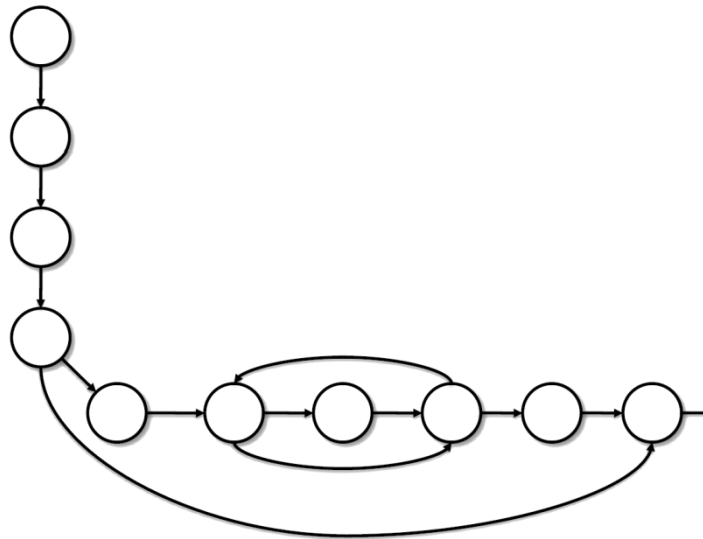
# Détection statique

- Identification de Graphe
  - **Définition** (graphe de flot de contrôle (GFC)): graphe orienté  $(N, E)$  avec  $N$  ensemble de sommets du graphe et  $E$  ensemble des arêtes. Chaque sommet est un bloc de base (« Basic Block »), i.e. une suite d'instructions consécutives se terminant par:
    - Soit une instruction de transfert;
    - Soit une instruction séquentielle suivie d'une instruction appartenant à un autre bloc de base.

Chaque arrête  $e$  issue d'un bloc de base est une sortie conditionnelle ou inconditionnelle de ce bloc.

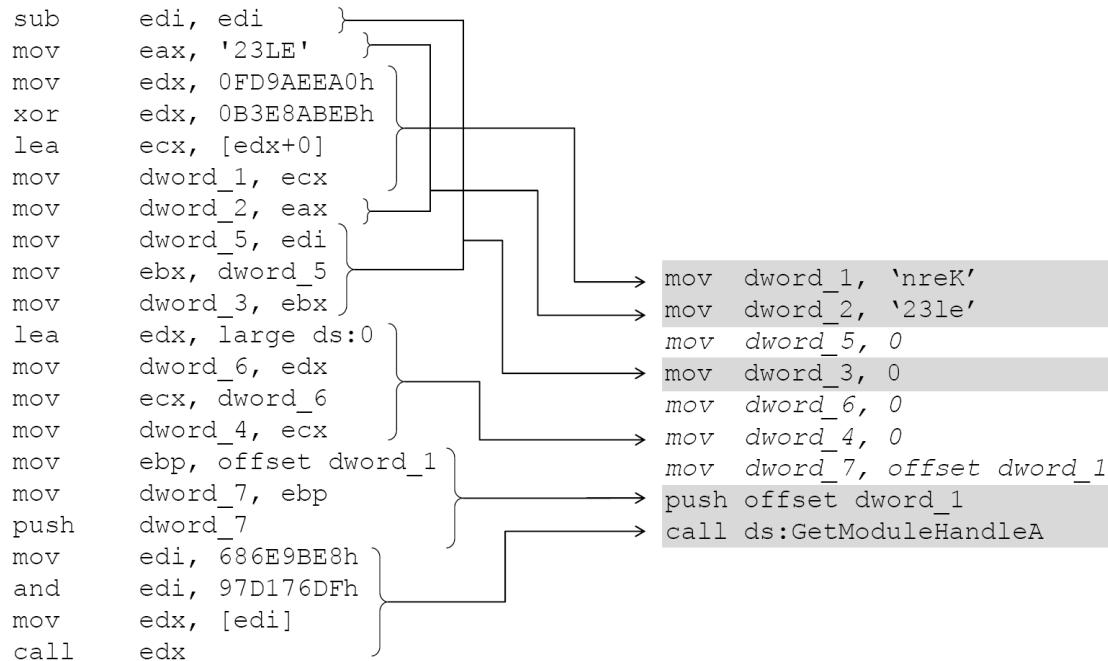
# Détection statique

- Identification du graphe de flot de contrôle
  1. **Génération du GFC**
  2. Normalisation des blocs
  3. Optimisation inter-bloc



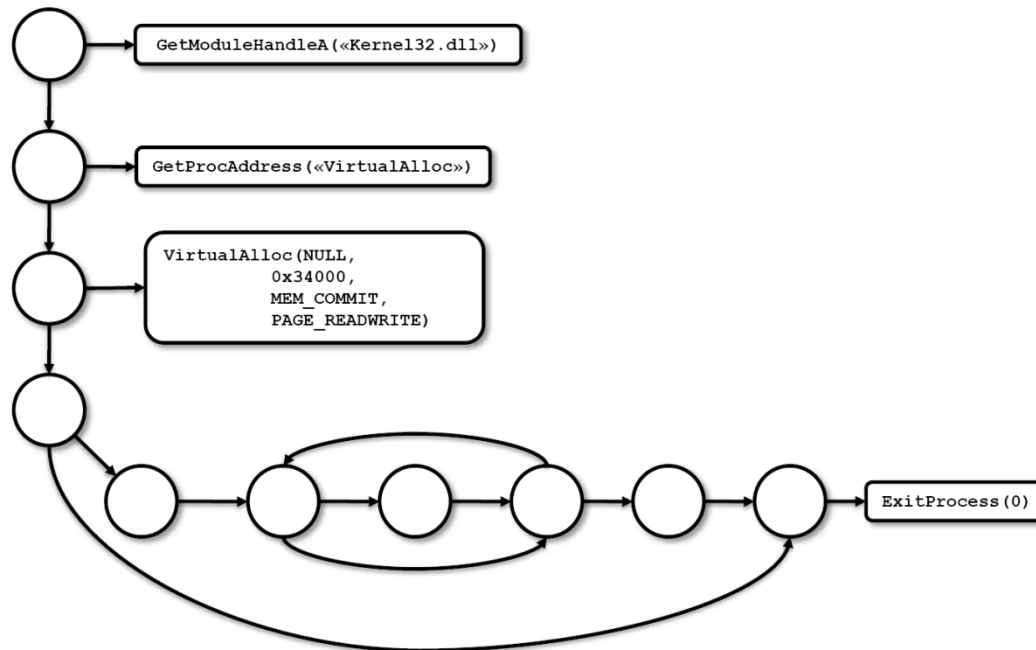
# Détection statique

- Identification du graphe de flot de contrôle
  1. Génération du GFC
  2. **Normalisation des blocs**
  3. Optimisation inter-bloc



# Détection statique

- Identification du graphe de flot de contrôle
  1. Génération du GFC
  2. Normalisation des blocs
  3. **Optimisation inter-bloc**





# Détection dynamique

- Nécessite un environnement d'exécution
  - **Sûr** pour empêcher l'action malveillante sur l'hôte
  - **Transparent** du point de vue du malware pour ne pas altérer son comportement
- Une approche de détection
  - Traitement des traces d'exécution
  - Suivi des API invoquées
  - Dépendances entre API

# Détection dynamique

- Instrumentation dynamique de binaires (DBI)
  - Permet d'instrumenter un programme comme on le souhaite pour observer son comportement au niveau des instructions ASM.
  - Ex: Pin, DynamoRIO, Valgrind, etc



Les DBI sont détectables

- ex: l'outil eXaït permet la détection de PIN de 15 façons différentes

# Détection dynamique

- Les machines virtuelles
  - Émulation
    - Simule un CPU, ex: Bochs
    - Permet d'émuler un CPU sur une autre architecture (ex: ARM sur x86)
  - Virtualisation
    - Exécute directement les instructions non privilégiés par translation de binaire, ex: VMWare, QEMU, VirtualPC, VirtualBox



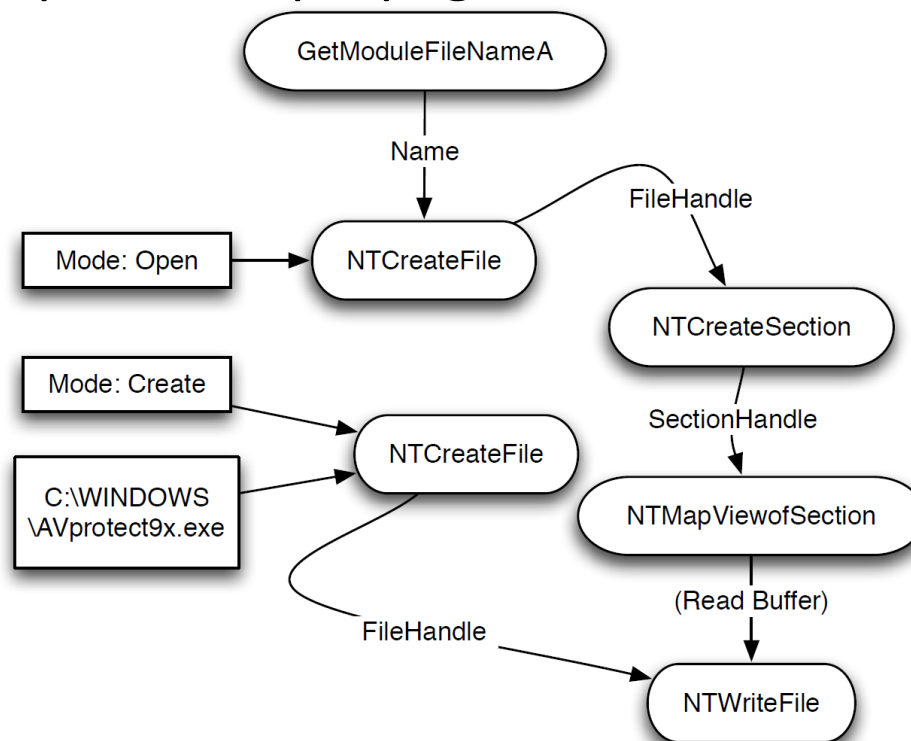
Les machines virtuelles sont détectables

# Détection dynamique

- La virtualisation matérielle
  - S'appuie sur un jeu d'instruction particulier apparu avec AMD-V et Intel-VT
  - Permet de filtrer les instructions privilégiées, les entrées/sorties et les accès à la mémoire
    - Xen : hyperviseur open source pour linux
    - Ether: outils basé sur Xen dédié à l'analyse de malware
      - EtherUnpack : analyse pas à pas
      - EtherTrace : monitoring d'appels système

# Détection dynamique

- Détection comportementale
  - Signature « comportementale »
    - Ex: réplication, propagation, résidence, etc.



# Détection dynamique

- Difficultés lié à l'environnement
  - Test de connectivité internet
  - Droit utilisateur (Admin/User)
  - Services ciblés (exploitation d'un programme spécifique)
  - Données environnementales (présence d'un fichier particulier)
  - Bombes logiques (action malveillante à une date donnée)
  - etc

# Détection dynamique : sandbox

- Cuckoo sandbox, opensource
  - Framework extensible (écrit en python)
  - VBox, KVM, VMWare, Xen
- JoeSecurity, propriétaire avec version basique (gratuite) et pro
  - Machines virtuelles et physiques

# Panorama des malwares

Chevaux de Troie,  
Virus / Vers  
Rootkits / Bootkits  
Botnets  
Ransomware

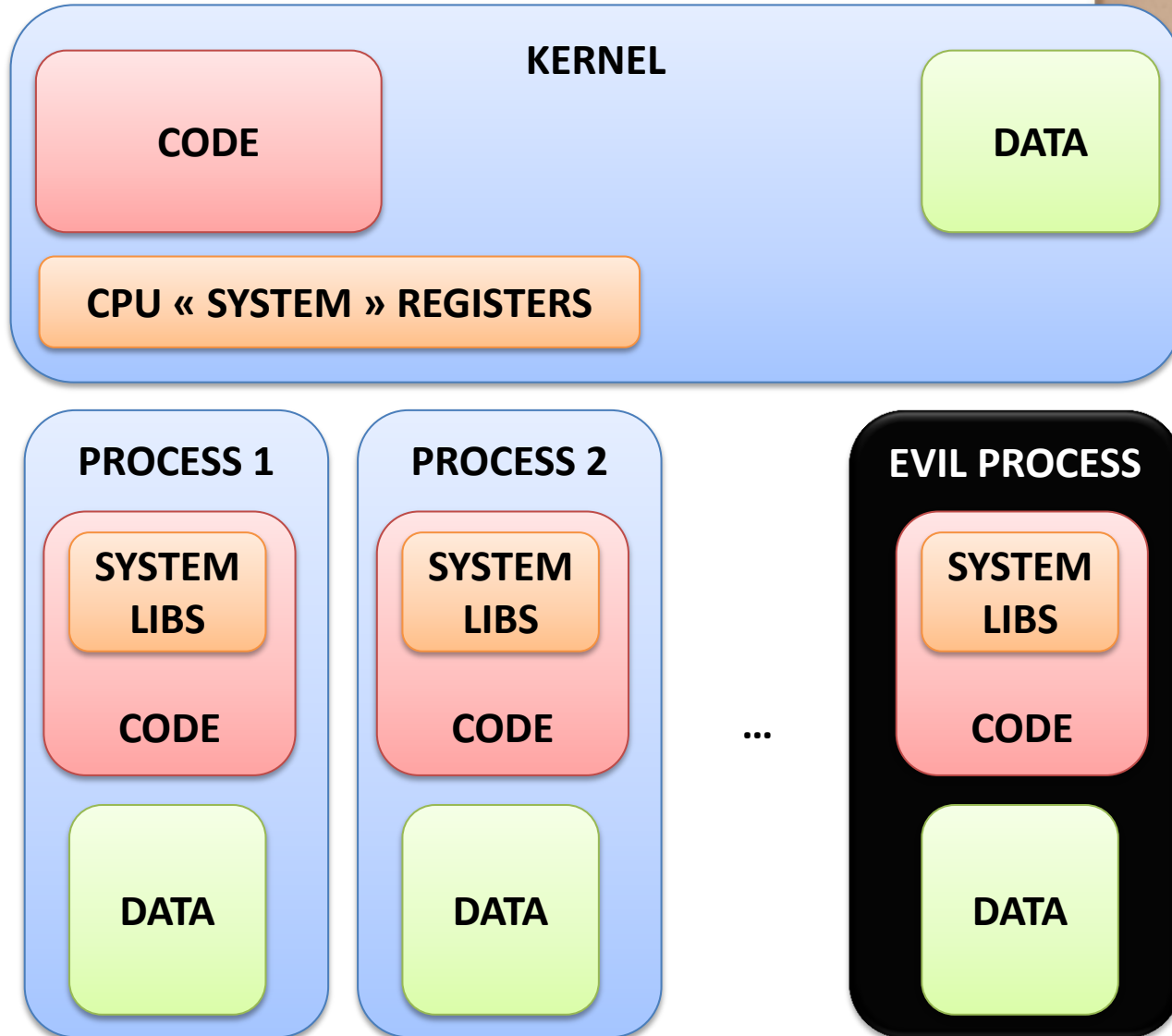


# Chevaux de Troie



- Définition :
  - Malware d'apparence légitime, conçu dans le but d'exécuter des actions malveillantes à l'insu de l'utilisateur
- Fonctionnement:
  - Il s'agit d'un programme simple sans technique spécifique

# Chevaux de Troie



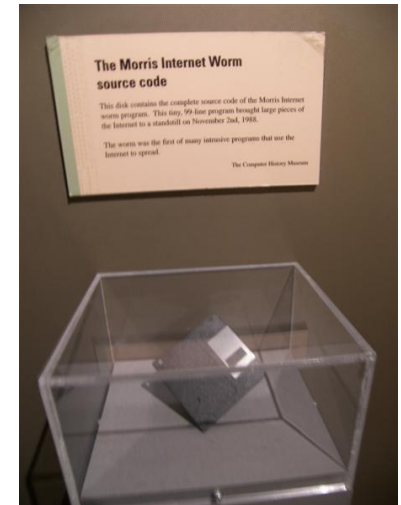
# Vers



- Définition :
  - Malware auto-reproducteur se propageant au moyen du réseau
- Fonctionnement:
  - différentes techniques assurant la reproduction:
    - Pièce jointe d'un mail (« mailer »)
    - Exploitation d'une vulnérabilité sur une cible
    - Etc.
- Prévention : Firewall
  - Limiter les services ciblés (historiquement SMB, RPC, etc)
  - si le vers ne mute pas, détection du code envoyé qui est le même que l'original dans le cas d'un « mailer »

# Vers

- Exemples célèbres : **Morris (1988)**
  - 1<sup>er</sup> ver connu conçu pour se propager uniquement
  - Exploitation de:
    - Mode DEBUG de sendmail
    - Buffer overflow dans finger
    - Mots de passe utilisateur faibles
  - Création du premier CERT  
(Computer Emergency Response Team)
  - Condamnation de Robert Morris à 400h de travaux d'intérêts publics et plus de 10'000\$ d'amende

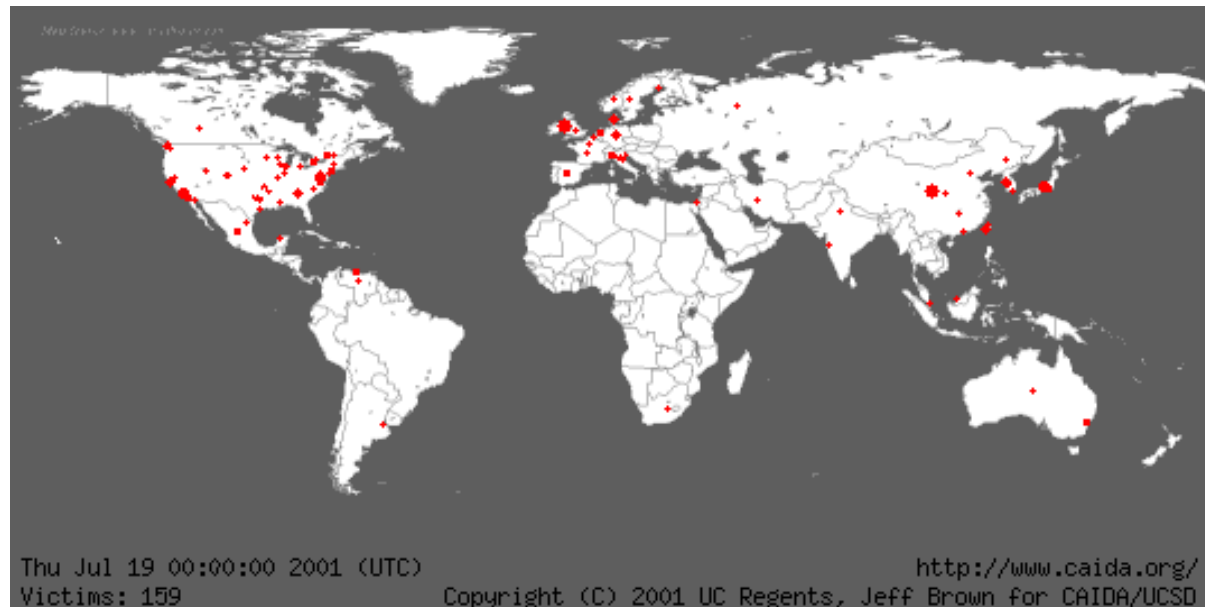


# Vers

- Exemples célèbres : **I love You (2000)**
  - Pièce jointe d'un mail Love-Letter-for-you.txt.vbs
  - Actions:
    - Modifie la page de démarrage d'IE pour télécharger un cheval de Troie (WIN-BUGSFIX.exe)
    - Modifie le registre pour s'exécuter automatiquement
    - Remplace certains fichiers par une copie de lui-même
    - Propagation via les contacts outlook et le client IRC mIRC
  - 3,1 millions de machines infectés
  - Auteur présumé : Onel Guzman (philippin 24ans), relâché sur le champ car aucune loi contre le hacking à l'époque

# Vers

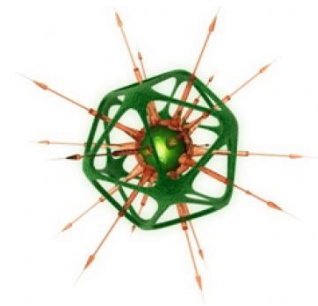
- Exemples célèbres : **Code Red (2001)**
  - Exploite une vulnérabilité dans les servers IIS
  - Lancement d'une attaque de type DoS
  - 359 000 machines infectées en 24h



# Vers

- Exemples célèbres : **Conficker (novembre 2008)**
  - Exploitation de la faille MS08-067 (RPC) permettant d'exécuter de code à distance en temps que SYSTEM
  - Plusieurs millions de machines infectées
  - Utilisation d'un AutoRun pour propagation via clé USB
  - Actions:
    - Désactivations de Windows Update, centre de sécurité et Windows Defender
    - Connexion à des serveurs de commandes et de contrôle (C&C) :
      - Collecte d'information
      - Téléchargement de charges supplémentaires
      - etc

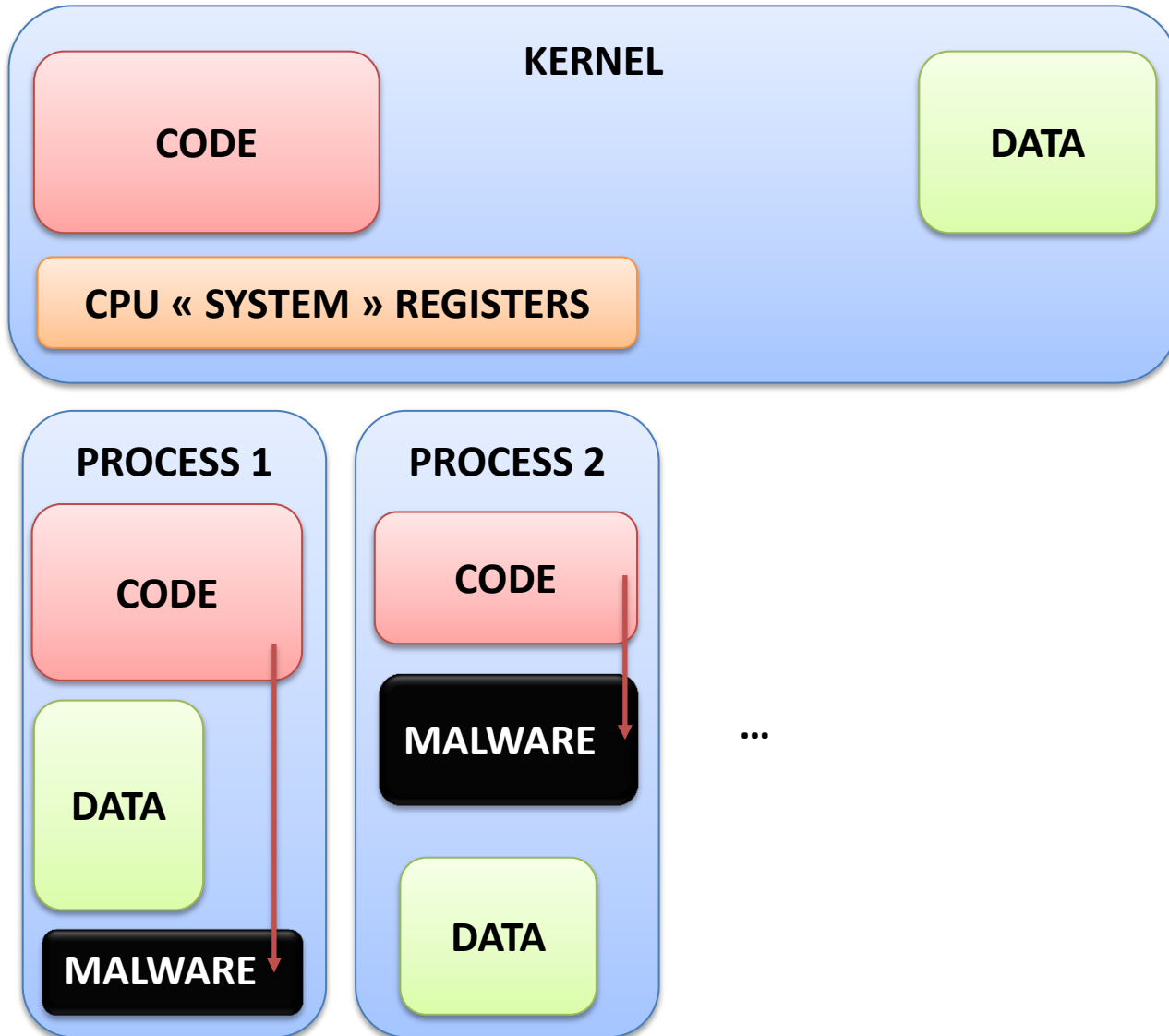
# Virus



- Définition :
  - Malware auto-reproducteur se propageant en infectant un programme hôte
- Fonctionnement:
  - Modification du flow de control de l'hôte pour s'exécuter:
    - Point d'entrée du programme (Entry Point Obfuscation)
    - Fin du programme
    - À un endroit précis dont le virus est sûr de l'exécution (ex : modification de l'IAT)



# Virus



# Virus

- Exemples célèbres: **Tchernobyl (ou CIH 1997)**
  - Virus particulièrement destructeur
    - Écrasement du 1<sup>er</sup> Mo de chaque disque dur (et donc du MBR)
    - Tentative de flashage du BIOS
  - Auteur du virus arrêté le 29 avril 1999 et relâché sur le champ car aucune plainte déposée !
  - Cible Windows 95, 98 et ME

# Virus

- Exemples célèbres: **Win32.MetaPHOR (2002)**
  - Virus métamorphe
  - 14000 lignes d'assembleur x86
  - 90% du code correspond au moteur de métamorphisme
  - Exécution lorsque l'hôte se termine
  - Particulièrement difficile à détecter

# Virus

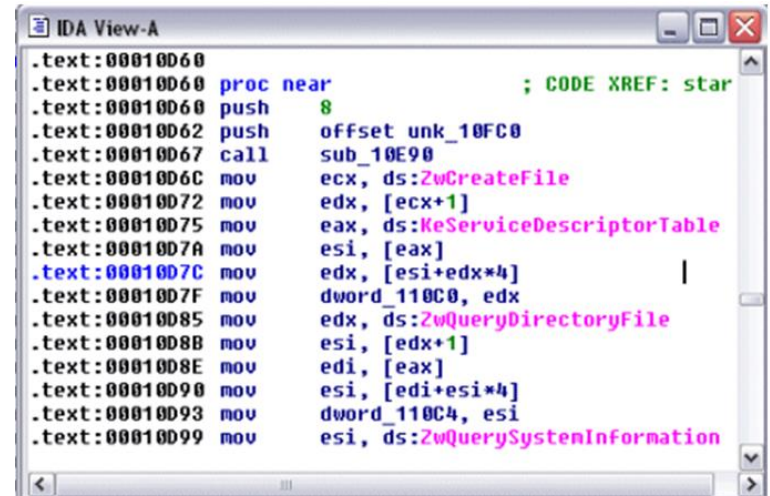
- Techniques de prévention:
  - Détection du test de surinfection
  - Vérification d'intégrité
  - Empêcher la modification de binaires
  - Politique de signatures d'exécutables

# Rootkits : définition

- Définition :
  - Malware permettant de modifier le comportement d'un système d'exploitation afin de cacher du code
- Fonctionnement:
  - Différents types de rootkits (*Joanna Rutkowska*)
    - Type 0 : interactions « documentées » avec le système
    - Type1 : modification des constantes du système
    - Type2 : modification des variables du système
    - Type3 : virtualisation du système

# Rootkits : exemple

- Exemples célèbres: **Rootkit SONY (2005)**
  - Rootkit dans des logiciels de gestion de DRM de SONY pour limiter la copie
  - Hooks de la SSDT
    - ZwCreateFile
    - ZwQueryDirectoryFile
    - ZwQuerySystemInformation
    - etc
  - Masque les fichiers, répertoires, clés de registre, processus commençant par « \$sys\$ »



The screenshot shows a window titled 'IDA View-A' displaying assembly code. The code is a function hook for the SSDT (System Service Descriptor Table). It starts with a 'proc near' directive and a comment '; CODE XREF: star'. The function pushes the offset 'unk\_10FC0' and calls 'sub\_10E90'. It then moves 'ecx' to 'ds:ZwCreateFile'. Next, it moves 'edx' to '[ecx+1]' and 'eax' to 'ds:KeServiceDescriptorTable'. It then moves 'esi' to '[eax]' and 'edx' to '[esi+edx\*4]'. It moves 'dword\_110C0' to 'edx' and 'ds:ZwQueryDirectoryFile' to 'edx'. It then moves 'esi' to '[edx+1]' and 'edi' to '[eax]'. It moves 'esi' to '[edi+esi\*4]' and 'dword\_110C4' to 'esi'. Finally, it moves 'esi' to 'ds:ZwQuerySystemInformation'.

```
.text:00010D60
.text:00010D60 proc near ; CODE XREF: star
.text:00010D60 push 8
.text:00010D62 push offset unk_10FC0
.text:00010D67 call sub_10E90
.text:00010D6C mov ecx, ds:ZwCreateFile
.text:00010D72 mov edx, [ecx+1]
.text:00010D75 mov eax, ds:KeServiceDescriptorTable
.text:00010D7A mov esi, [eax]
.text:00010D7C mov edx, [esi+edx*4]
.text:00010D7F mov dword_110C0, edx
.text:00010D85 mov edx, ds:ZwQueryDirectoryFile
.text:00010D8B mov esi, [edx+1]
.text:00010D8E mov edi, [eax]
.text:00010D90 mov esi, [edi+esi*4]
.text:00010D93 mov dword_110C4, esi
.text:00010D99 mov esi, ds:ZwQuerySystemInformation
```

# Rootkits : protection

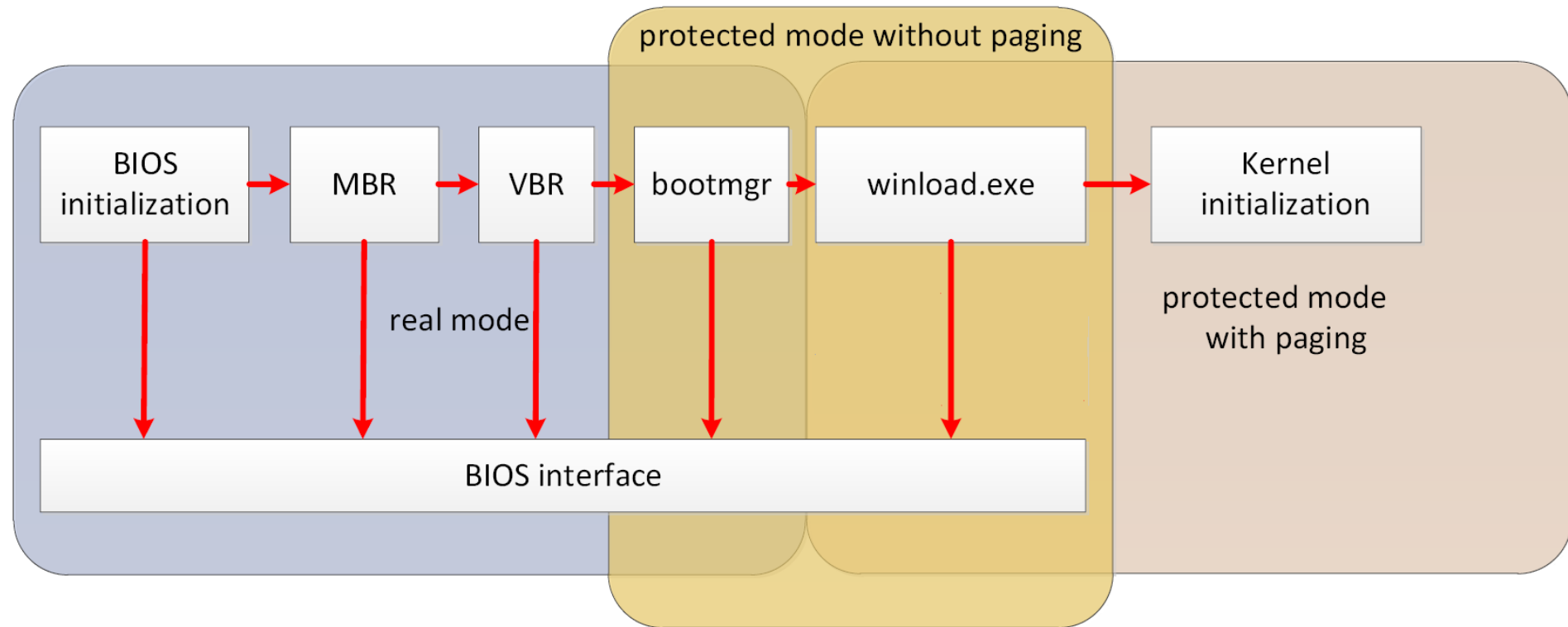
- Niveau kernel : patch guard (windows 64 bits)
  - IDT, GDT, SSDT, images systèmes (ntoskrnl.exe, ndis.sys, hal, etc), MSRs, etc
  - Fonctionne par contrôle d'intégrité
- Rootkit de niveau 3 **très** difficiles à détecter mais aucun connu actuellement
- Différents outils pour le userland:
  - Contrôle d'intégrité de fichiers
  - Corrélation TID/PID en cas de DKOM
  - RAW accès au système de fichier
  - etc

# Bootkits

- Définition :
  - Malware affectant le processus de boot (amorce) d'une machine afin de rester actif au sein du système d'exploitation
- Fonctionnement:
  - Hook d'un service du firmware (BIOS/UEFI) pour prendre le contrôle dans le flow d'exécution du boot (MBR, VBR, etc)



# Bootkits : fonctionnement



# Bootkits : exemple

- Académiques:
  - Stoned <http://www.stoned-vienna.com>
  - Vbootkit <http://www.nvlabr.in>
  - DreamBoot <https://github.com/quarkslab/dreamboot>
- Réels:
  - Win64/Olmarik (TDL4)
  - Win64/Olmasco (MaxSS)
  - Win64/Rovnix

# Bootkits : protection

- Secure Boot
  - Chaîne de confiance (vérification cryptographique) du matériel (composant TPM) jusqu'au système d'exploitation
  - Nécessite une coopération entre
    - Le matériel (TPM)
    - Le firmware (BIOS/UEFI)
    - L'OS
  - Utilisés sur les OS récents, smartphones compris

# Botnets



- Définition : Réseau de machines compromises (bot) permettant de mener des actions distribuées (spam, DOS, brute force, etc)
- Fonctionnement:
  - Communication vers un serveur de **C**ommandes et de **C**ontrôle (C&C)
    - Canaux IRC (historique)
    - P2P pour ne pas être centralisé
    - HTTP
    - etc

# Botnets

- Exemples célèbres : **Rustock (2006)**
  - 150 000 à 2 400 000 machines infectées
  - 30 milliards de Spams par jour
  - 250 000\$ promis par Microsoft pour l'identification des auteurs

# Botnets

- Exemples célèbres : **Waledac (2006)**
  - 1,5 milliards de Spams par jour (1% du spam mondial)
  - 277 noms de domaines pour l'envoi de spam
  - Communication via P2P
  - 70 000 à 90 000 machines infectées

# Ransomware



- Définition :
  - Malware bloquant l'ordinateur de la victime tant qu'une rançon n'a pas été payée
- Fonctionnement:
  - Les bloqueurs
  - Les crypto-ransomware

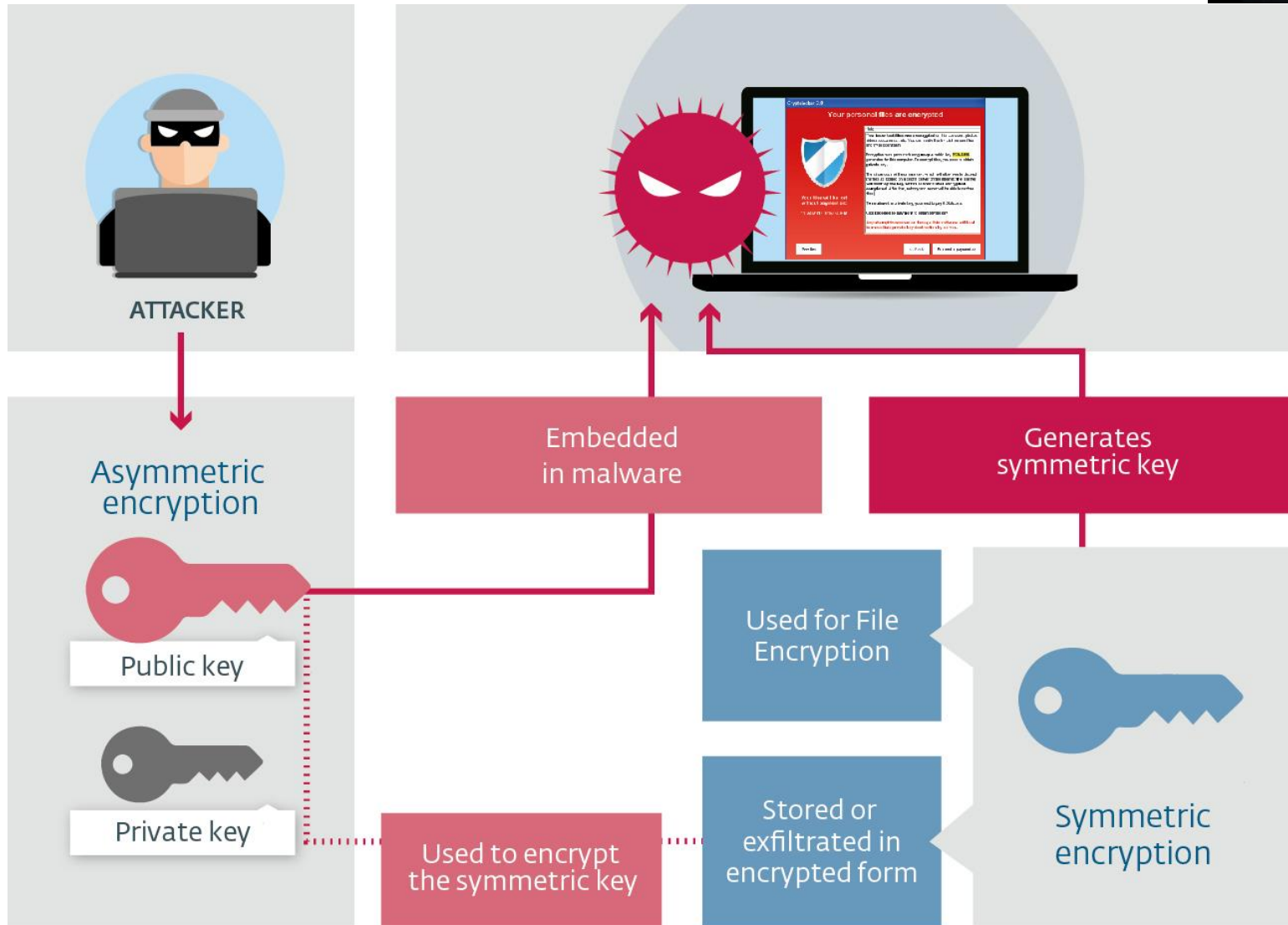
# Ransomware : fonctionnement



- Le ransomware embarque une clé publique asymétrique  $K_{pub}$  (ex: RSA)
- Cette clé  $K_{pub}$  permet de chiffrer une clé de session symétrique  $K_s$  (ex: AES) envoyée au serveur
- La clé  $K_s$  est utilisée pour chiffrer les fichiers de l'utilisateur
- Le seul moyen de récupérer les fichiers d'origine consiste à récupérer  $K_s$



# Ransomware : illustration



# Conclusion

- Difficile de modéliser les malware
  - Menaces complexe et très variés
- Évolution constante en fonction
  - Du matériel ex: virtualisation
  - Des OS : patchguard
  - Des firmware : bootkit
- Constitue un bonne veille technologique:
  - Exploits
  - Enjeux stratégiques
  - Etc.

# Références

- Szor, P. “The art of computer virus research and defense”, *Addison-Wesley Professional*, **2005**
- Eagle, C. “The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler”, *No Starch Press*, **2008**
- Sikorski, M. & Honig, “A. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software”, *No Starch Press*, **2012**
- Filiol, É. “Les virus informatiques : théorie, pratique et applications ”, *Springer*, **2009**
- Hoglund, G. & Butler, J. “Rootkits: subverting the Windows kernel ”, *Addison-Wesley Professional*, **2006**
- Intel 64 and IA-32 Architectures Software Developer’s Manual (Combines Volumes: 1,2A, 2B, 2C, 3A, 3B and 3C)