FDP/Winbagility

INTROSPECTION ET DÉBOGAGE FURTIFS DE VM

Sommaire

Objectifs

Débogage furtif de systèmes Windows

Aucune modification de la VM

- Eviter la détection (contrôle de code, chargement de driver,…)

Analyses de drivers malveillants

- Ring-0 contre Ring-0 => Difficultés
- Certains ne se chargent pas en mode /DEBUG

Analyse avec PatchGuard activé

- Eviter le BSOD

Analyse dynamique

- Analyse de malware
- Détection de compromissions

Instrospection

FAST DEBUGGING PROTOCOL

Introspection

Reconstruction de l'état de l'invité

- Depuis l'hyperviseur
- Vue complète du système
- Sans artefact dû à un agent

Isolé du système analysé

- Au chaud, à l'abri d'un driver malveillant
- Et des modifications de code !

Possibilité de poser des points d'arrêt

- Modification du comportement de l'invité
  - Empêcher la création d'un fichier
  - Empêcher l'établissement d'une connexion
  - Eviter la lecture d'une zone mémoire

Etat de l'art

Analyse dynamique :

- Agent qui débogue un processus
- Agent qui *hook* ntdll.dll et/ou kernel32.dll
- Agent qui *hook* la SSDT
- Techniques facilement détectables

LibVMI :

- Utilise un hyperviseur
- Support de Xen et KVM
- Hôte Linux uniquement
- Point d'arrêt furtifs (mais peu performants)

Voyons quelques notions de virtualisation…

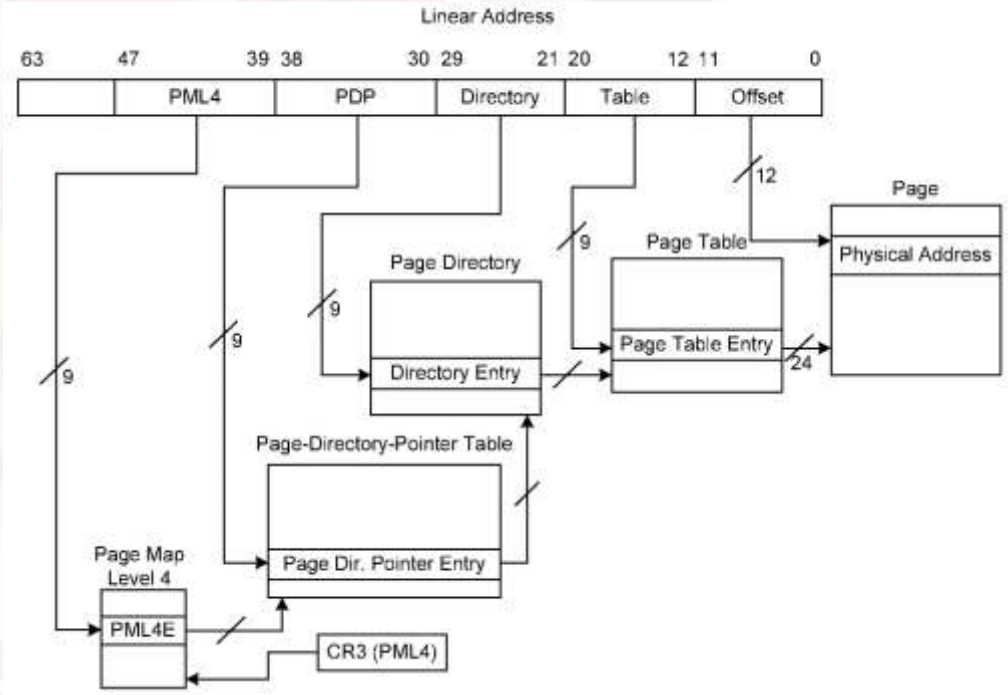
Page Table

Trucs cools:

- Protection des pages du noyau
- Protection des accès entre processus
- Isole les processus

Trucs moins cools:

- Visible pour le noyau
- Modifications détectables

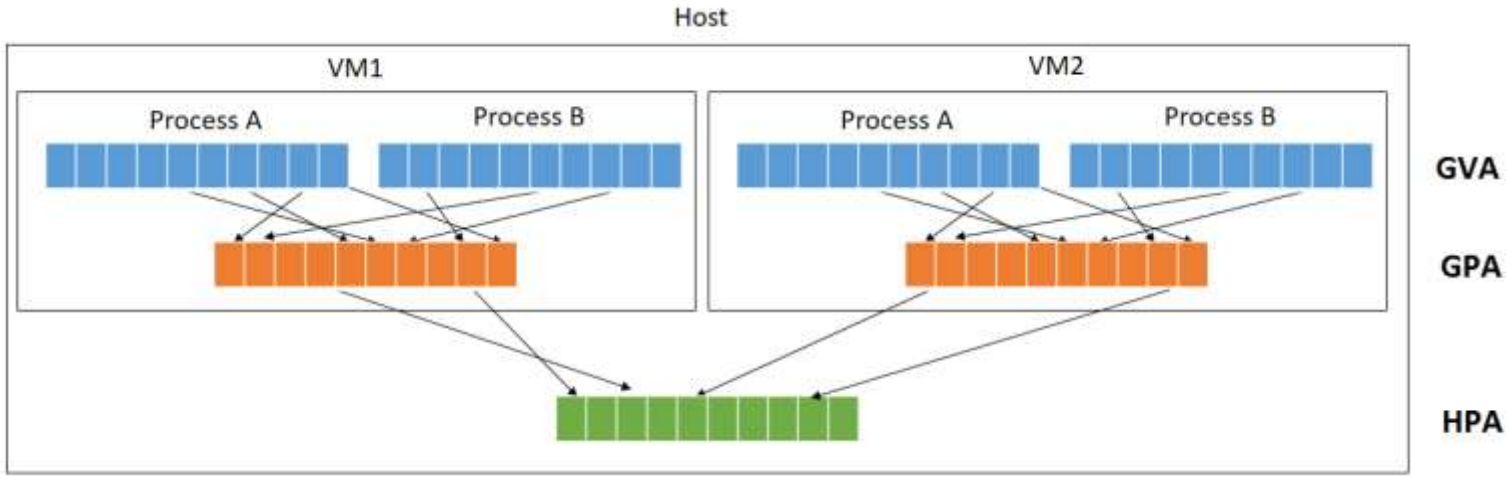


Extended Page Table

Extended Page Table

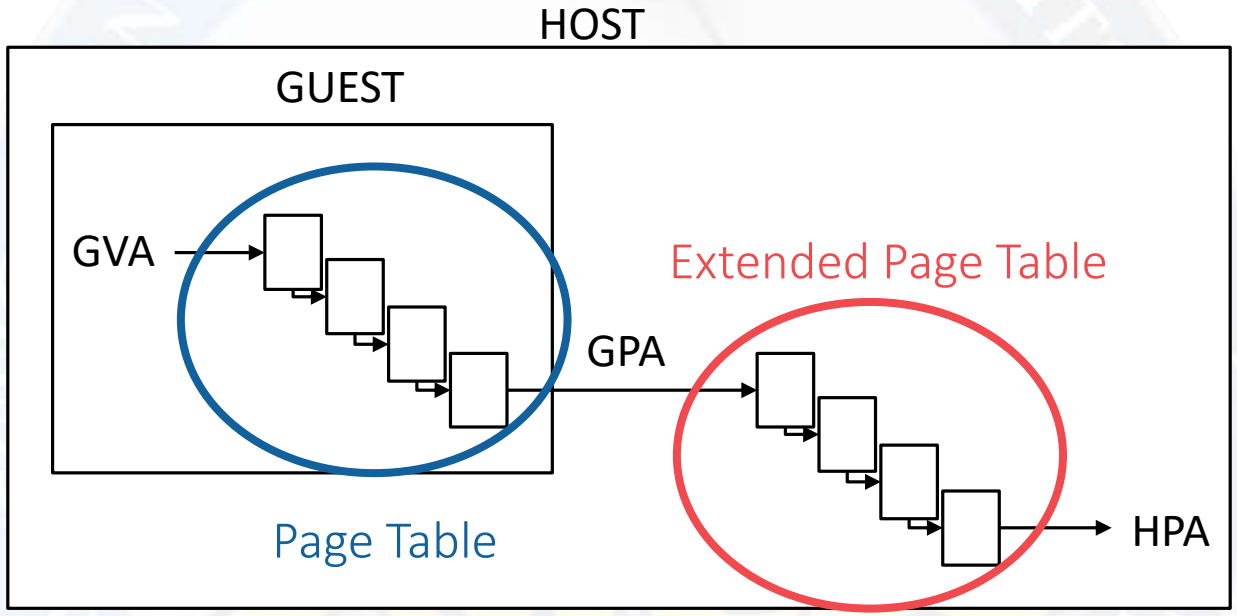
Trucs cools :

- Mécanisme de translation presque identique à la MMU
- Protection de la mémoire hyperviseur
- Isole les machines virtuelles
- eXecute only



Extended Page Table

Vue globale :



Trucs trop cools :

- Invisible et transparent pour l'invité
- Manipulable depuis l'hôte

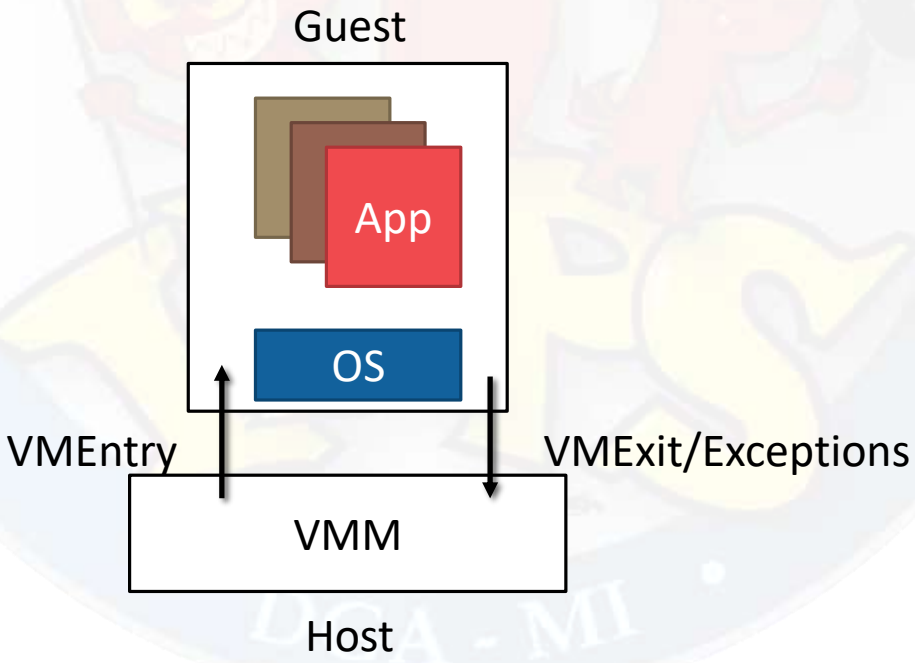
VMEntry/VMExit

Les VMEntry lancent l'exécution de l'invité

- L'invité est le chef du processeur sauf que…

Les exceptions repassent le CPU en mode « Hyperviseur »

- Et là, l'hôte est le chef de l'invité !
- Exécution d'un *Exception Handler*



Exceptions

Monitor Trap Flag

- Chaque instruction exécutée lève un #MTFException
- Permet de faire du Single-Step de l'invité

MovDRx

- Exception lors d'un accès au Debug Register

HLT

- Instruction 0xF4
- #HLTException en Ring-0
- #GPException en Ring-3

EPT Violation

- En cas d'erreur lors d'un accès mémoire
- Ex.: Exécution d'une page non exécutable

BreakPoints Classiques

SoftwareBreakpoint

- Papa, maman ? Comment on fait les 0xCC ?
- Modification du code
  - BSOD (CRITICAL_STRUCTURE_CORRUPTION)
- IDTR 3 inutilisable en R0 sans /DEBUG
  - BSOD (KMODE_EXCEPTION_NOT_HANDLED)
- Facilement détectable (Contrôle du code,…)

HardwareBreakpoint

- Utilisation des Debug Registers
- IDTR 1 inutilisable en R0 sans /DEBUG
  - BSOD (KMODE_EXCEPTION_NOT_HANDLED)
- Tout aussi détectable (DR7 != 0x400, DR0-3 != 0)

Il faut des HyperBreakpoints !

PageHyperBreakpoint

Installation du point d'arrêt

- Retrait de certains droits d'une GPA sur une HPA
  - Ex.: Retrait de X pour un point d'arrêt en exécution
- Invalidation de la GPA (INVEPT)

Pseudo-Code du *Nested Page Fault Handler* :

```
Si IsBreakpoint(FaultingAddress, AccessType)
 Alerter le débogueur
FinSi
Définir les permissions de FaultingAddress à RWX
Invalider FaultingAddress
Activer le MFT
SingleStep
Désactiver le MFT
Restaurer les permissions altérées de FaultingAddress
Invalider FaultingAddress
```

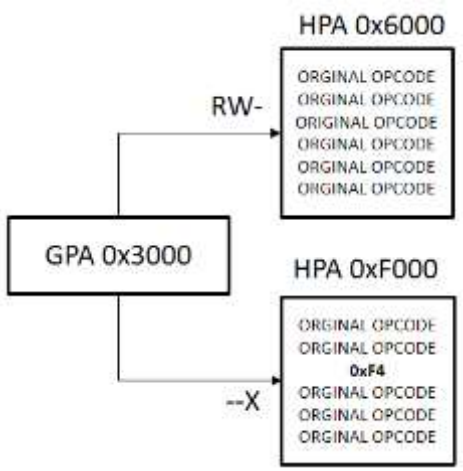
SoftHyperBreakpoint

Installation du breakpoint

- Allocation d'une Host Physical Page
- Copie de la Host Physical Page et installation d'un HLT (0xF4)

Pseudo-Code du #EPTException Handler

```
Si AccessType == Lecture/Ecriture
   Définir la HPA de FaultingAddress à OriginalHPA
   Définit la HPA en RW
Sinon
   Définir la HPA de FaultingAddress à ModificatedHPA
   Définit la HPA en X
FinSi
Invalider FaultingAddress
```



Pseudo-Code du #HLTException/#GPException Handler

```
Si IsBreakpoint(Guest.Rip)
   Alerter le débogueur
Sinon
   Traitement normal
FinSi
```

HardHyperBreakpoint

Utilisation des Debug Registers de l'invité

#DB Exception Handler

```
Si Breakpoint
    Alerter le débogueur
    Retirer l'interruption
Sinon
    Passer l'interruption à l'invité
FinSi
```

#MovDRx Exception Handler

```
Sauvegarde des DR Invisibles
Restauration des DR Visibles
Single-Step avec MTF
Sauvegardes des DR Visibles
Restauration des DR Invisibles
```

Remplacement des points d'arrêt de l'invité

- PageHyperBreakpoint
- Injection INT1 dans l'invité

Fast Debugging Protocol

API d'introspection et de débogage

- C
- Bindings Python
- Minimaliste
- Performante
- Intégrée à VirtualBox (5.0.18)

VirtualBox ?

- Cross-Platform (~~KVM~~, ~~XEN~~)
- Open Source (~~VMware~~)

Permet de :

- Lire/Ecrire de la mémoire virtuelle ou physique
- Lire/Ecrire des registres et MSR
- Injecter des interruptions
- Exécuter une instruction (Pas à pas)
- Pause/Resume et Save/Restore la machine virtuelle
- Positionner des points d'arrêt

Performances de FDP

Pas à pas

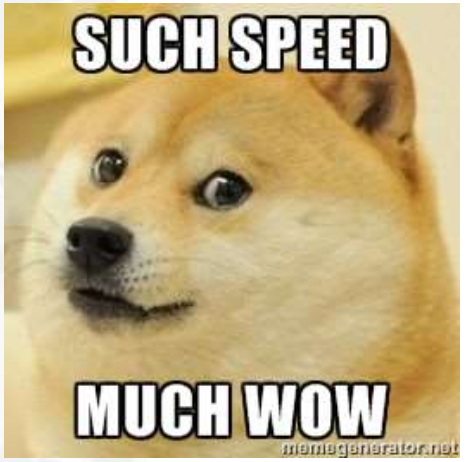
- ~105000 Instructions/s

Lecture/Ecriture

- ~600000 Lectures virtuelles de 4K/s => 2,3 Go/s
- ~800000 Lectures physiques de 4K/s => 2,9Go/s

Restauration

- VM de 2Go mémoire en 2sec



Points d'arrêt

Type	Hits/s
HardHyperBreakpoint	27300
SoftHyperBreakpoint	20900
PageHyperBreakpoint	7100

Démos !

Winbagility

DÉBOGAGE FURTIF DE WINDOWS

Etat de l'art

Débogage :

- /DEBUG + WinDbg
  - Non furtif (/DEBUG)
- VirtDbg
  - Couteux et difficile d'utilisation (PCB, FPGA, machine physique…)
  - Non furtif (0xCC, modification des Page Table, injection de driver)
  - GDB seulement
- LibVMI+VMIDbg
  - Furtif (PageHyperBreakpoint)
  - GDB seulement
  - Linux et Xen
  - Performances moyennes
  - Pas de support de WinDbg

Windows en mode « STOCK »

Débogueur inactif

Vérification de la signature des drivers

PatchGuard veille !

- Détection des modifications
  - Du code
  - Des structures/MSR critiques (IDT, LSTAR…)
  - Des LAPIC
  - …
- Difficile de faire un débogueur compatible

Nt!KdDebuggerDataBlock (KDBG)

- Essentiel pour WinDbg
- Chiffré (Windows 8+)

Support de Winbagility

Windows

- * x64

WinDbg/IDA Pro

- Implémentation d'un serveur KD générique
- Interface de 12 fonctions

STUB

- CRASH (Dump Physique)
- GDB (VMware, non furtif,..)
- FDP :D
- LIVEKADAY (WinPmem)

Analyse Initiale

Déchiffrer nt!KdDebuggerDataBlock

- Trouver nt!KiDivideErrorFault (IDTR ou KPCR.IdtBase)
- Calculer les offsets (Brute force de PDB)
  - nt
  - nt!KdVersionBlock,
  - nt!KiWaitAlways…
- Récupérer les 3 clefs
- Essayer de déchiffrer nt!KdDebuggerDataBlock

Récupérer des informations de base pour WinDbg

- KPCR, KPRCB, PsLoadedModuleList…

Profits !

Extensions WinDbg

Permet d'utiliser pleinement FDP avec WinDbg

!hba

- Interface pour les PageHyperBreakpoint

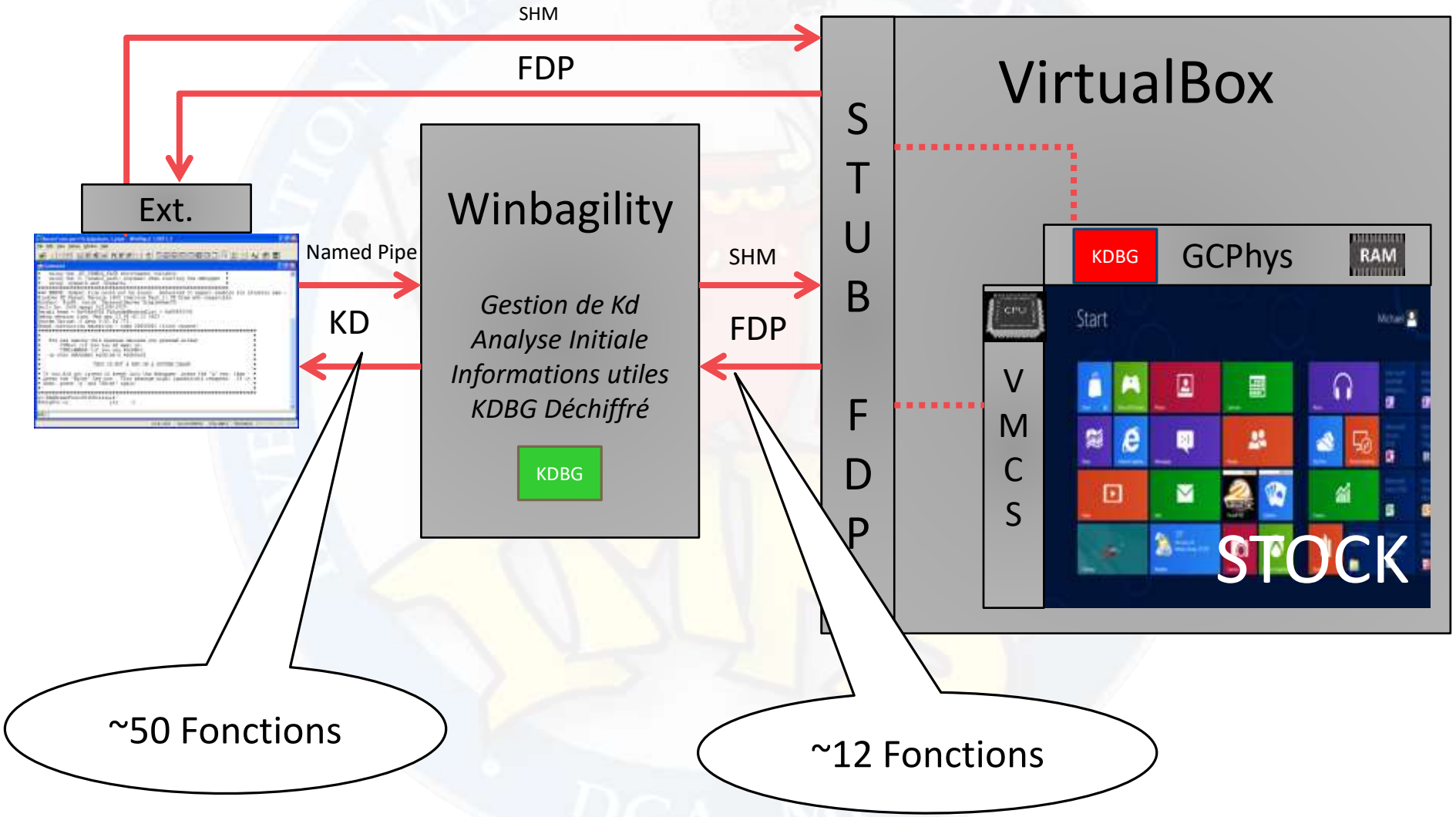
!save & !restore

- Restauration du contexte complet (registres, disques, …)
- Facilite la répétition des analyses après compromission

!hbint

- Injection d'interruption
- Ex. : Simulation d'une faute de page

Architecture avec FDP



Démos !

Limitations

Ne supporte qu'un CPU

Breakpoints sur page physique

- Breakpoints valides pour tous les processus
- Supervision des modifications des PTE

Virtualbox ne gère pas la virtualisation imbriquée

- Analyse de Ramooflax impossible… (Stub GDB+VMWare)

Pagination

- Gestion de pagefile.sys
- FDP_ReadPhysicalDisk ?

Noyau non collaboratif

- .process /p /i ...
- Exceptions non gérées
- Pas de log (DbgPrint)

Serveur KD non exhaustif

Evolutions

Gestion de x86 pour FDP et Winbagility (En cours)

Portabilité Linux (FDP)

Gestion d'évènements (MovCrX, CpuId…)

Fiabilisation du mode multi-CPU

- Gérer les *race condition* sur les Extended Page Table…

Amélioration des performances

- *Memory mapping* (mémoire physique invité, VMCS…)
- Points d'arrêt toujours plus performants
  - Conditions traitées par l'hyperviseur
  - Nouveaux types de points d'arrêt

Merci !

Questions ?

C'est dans les actes…

Open-Source ?

- Oui ! (http://winbagility.github.io/)
- Label1 : *Pull Request* bienvenus

Ca fonctionne sur/pour Linux ?

- L'invité oui !
- L'hôte non, Goto Label1

FDP à distance ?

- Goto Label1

Support de Windows XP ?

- Pas testé… mais normalement oui

Support de Windows 3.1 ?

- Mmmm…

FDP/Winbagility est-il un cybergiciel ?

- Mais oui, c'est clair !

Exemples d'introspection

Vérification de structures critiques
- SSDT, IDT, CI!g_CiOptions

Intégrité de code noyau
- Détection de modification (trampolines,…)

Analyses d'appels systèmes critiques
- NtOpenFile/NtCreateFile
- NtOpenProcess/NtCreateProcess
- NtDeviceIoControlFile…

Liste de processus, pilotes, …
- Liste « noire »
- Liste « blanche »

HyperBreakpoints

Tableau récapitulatif:

Type	Avantages	Inconvenients
PageHyperBreakpoint	Lecture/Ecriture/Exécution	Overhead Adresse Physique
SoftHyperBreakpoint	Overhead	Exécution Adresse Physique
HardHyperBreakpoint	Adresse Virtuelle Overhead Lecture/Ecriture/Exécution	Limités à 4

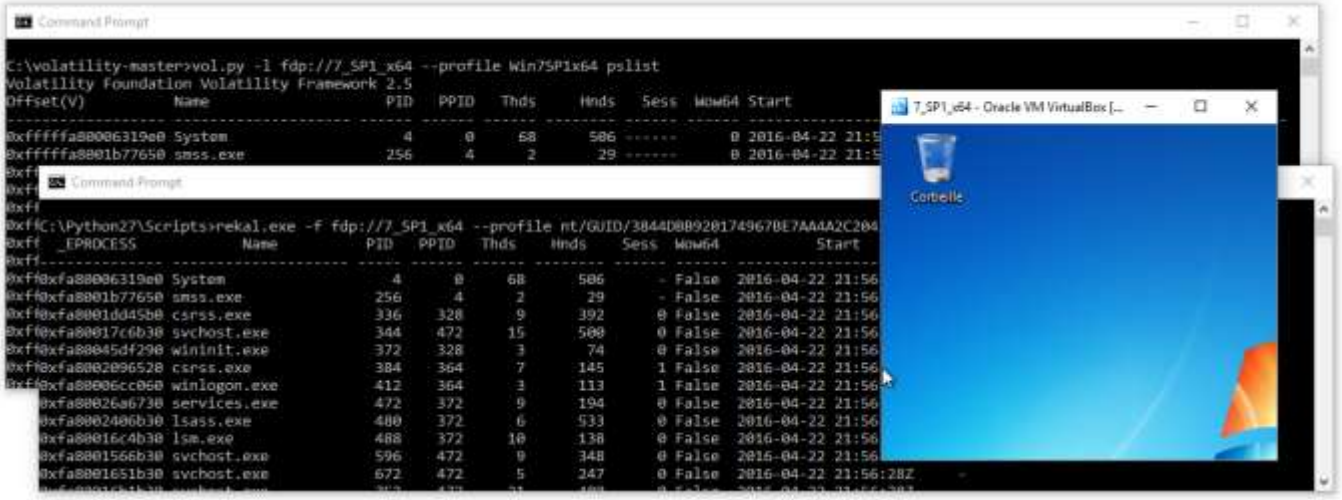
Bindings Python

Analyse d'appels systèmes

```
fdp = FDP("10_x64")
fdp.Pause()
KiSystemCall64 = fdp.ReadMsr(CPU0, MSR_LSTAR)
fdp.SetBreakpoint(CPU0, FDP_SOFTHBP, 0, FDP_EXECUTE_BP, FDP_VIRTUAL_ADDRESS, KiSystemCall64, 1)
fdp.Resume()

while True:
    if fdp.WaitForStateChanged() & FDP_STATE_BREAKPOINT_HIT:
        #DO STUFF
        fdp.SingleStep(CPU0)
        fdp.Resume()
```

« Address Space » Volatility & Rekall



VMCS

Espace mémoire de 4K

Définit un vCPU d'une machine virtuelle

Accessible depuis l'hôte

Contient:

- **Guest State Area** : Registres de l'invité
- **Host State Area** : Registres de l'hôte
- **VM-Execution Control Field** : Configuration des exceptions
- **VM-Entry Control Field** : Interruption en attente
- **VM-Exit Control Field** : Exception qui a généré le VMExit
- **VM-Exit Information Field** : Informations concernant le VMExit

KD Protocol

Protocole de débogage noyau
- WinDbg<->Kernel
- Server KD en ring 0
- Actif en mode /DEBUG

Analysé partiellement jusqu'à KD10
- ReactOS et VirtualKD

Gestion des pertes (ACK), Checksum, Resynchro…

Exemples de pkt:
- ReadPhysicalMemory
- SwitchProcessor
- SearchMemory
- SetContext
- …

Utilisable par:
- WinDbg
- IDA Pro

Architecture d'analyse de KD