Dissection de l'hyperviseur VMware

Brice L'HELGOUARC'H

6 Juin 2019

Motivations (1)

Virtualisation : quelle confiance ?

- Confiance « aveugle »
  - Analyse de malwares
  - Infrastructures critiques
  - *Cloud computing*
- Risque principal : l'isolation n'est plus **physique** mais **logicielle**

Motivations (1)

AMOSSYS

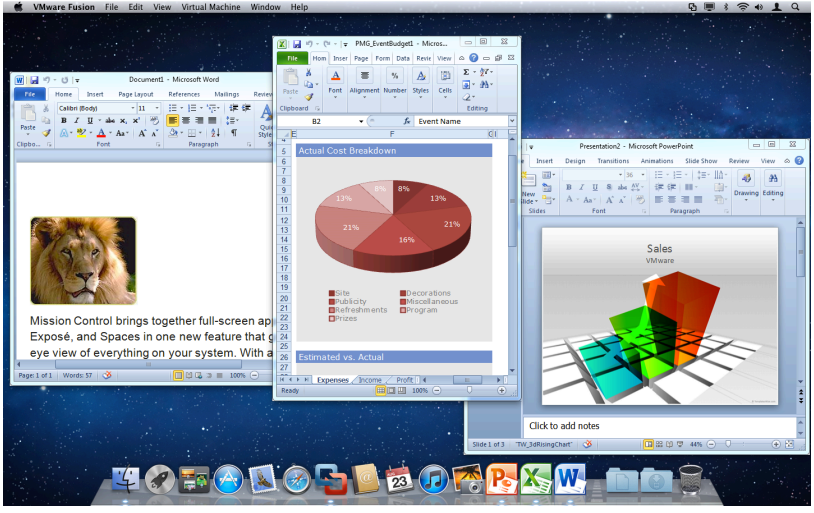
Virtualisation : quelle confiance ?

- Confiance « aveugle »
  - Analyse de malwares
  - Infrastructures critiques
  - *Cloud computing*
- Risque principal : l'isolation n'est plus **physique** mais **logicielle**

Cas de l'hyperviseur VMware

- Hyperviseur très répandu : Workstation (*desktop*), ESXi (serveur)
- Facilitation des interactions utilisateur
  - *Snapshots*
  - Accélération 3D
  - Mode Unity (Windows, macOS)

Motivations (2)



Fonctionnement général des hyperviseurs

Prérequis formels (1)

- Popek et Goldberg, *Formal Requirements for Virtualizable Third Generation Architectures*, 1974 : ensemble de conditions suffisantes (mais non nécessaires !) à la virtualisation sur une architecture
- Spécification des propriétés attendues d'un hyperviseur :
  - Fidélité,
  - Performance,
  - Contrôle sur les ressources.

Définition (instruction privilégiée)

Une instruction est dite *privilégiée* si l'exécution de celle-ci lance une exception lorsque le processeur n'est pas en mode superviseur.

Prérequis formels (2)

AMOSSYS

Définition (instruction sensible)

Une instruction est dite *critique* ou *sensible* si celle-ci interagit avec le matériel.

Théorème

Pour tout ordinateur de troisième génération conventionnel, un hyperviseur peut être construit si l'ensemble de ses instructions sensibles est un sous-ensemble de ses instructions privilégiées.

Prérequis formels (2)

AMOSSYS

Définition (instruction sensible)

Une instruction est dite *critique* ou *sensible* si celle-ci interagit avec le matériel.

Théorème

Pour tout ordinateur de troisième génération conventionnel, un hyperviseur peut être construit si l'ensemble de ses instructions sensibles est un sous-ensemble de ses instructions privilégiées.

- Processeurs x86_32 : instructions sensibles mais non privilégiées
- Contrôle sur les ressources difficile
- x86_32 n'est pas donc pas virtualisable **au sens de Popek et Goldberg**

Virtualisation assitée par le matériel (1)

AMOSSYS

Instructions VT-x

- Permet à l'hyperviseur d'intercepter automatiquement les instructions sensibles
- Deux nouveau modes d'exécution (*VMX root*, *VMX non-root*)
- Utilisation de *Virtual Machine Control Structures* pour contrôler les transitions entre ces deux modes

Virtualisation assitée par le matériel (1)

AMOSSYS

Instructions VT-x

- Permet à l'hyperviseur d'intercepter automatiquement les instructions sensibles
- Deux nouveau modes d'exécution (*VMX root*, *VMX non-root*)
- Utilisation de *Virtual Machine Control Structures* pour contrôler les transitions entre ces deux modes

Traitement des *VM exits*

- Contenu de la VMCS
  - État du système invité,
  - État du système hôte,
  - Champs d'information de sortie de la VM...
- Lorsqu'une instruction sensible est piégée dans l'hyperviseur :
  - Lecture des champs appropriés de la VMCS,
  - Exécution du traitement approprié et reprise de l'exécution de la VM.

Virtualisation assitée par le matériel (2)

```
vmxon
init VMCS
vmlaunch

while(1) {
   exit_code = read_exit_code(VMCS);
   switch(exit_code) {
      case TRIPLE_FAULT_EXIT:
         // Traitement en cas de triple fault
      case IO_INSTRUCTION:
         // Traitement en cas d'instruction I/O
      case VMXOFF_INSTRUCTION:
         // Traitement la VM exécute l'instruction vmxoff
      [...] // Autres exit codes
   }
   vmresume
}

vmxoff
```

Étude des mécanismes de l'hyperviseur VMware

Communication entre VM et hyperviseur : Backdoor

- Communication par le biais de ports I/O spéciaux (*VM exits*)
- Ports I/O 0x5658 (*slow*) et 0x5659 (*fast*)
- Activé par défaut
- Pas besoin d'être en ring 0 (choix d'architecture)
- Pas de différenciation du niveau de privilège de l'appelant

```
mov rax, 0x564D5868 ;magic number
mov rbx, 0x8686 ;params commande
mov rcx, 0x01 ;num commande
mov dx, 0x5658 ;port

in rax,dx
```

Listing 1 – Exemple de communication par Backdoor: récupération de la fréquence du CPU.

Protocoles sur Backdoor

RPCI

- Messages du système invité vers l'hyperviseur
- Informations sur le système invité, logs, drag n' drop, presse-papiers du guest, Unity...
- Exemple de commande : `info-set guestinfo.ip 192.168.122.123`
- Format des réponses `0/1`
- Utiliser l'outil `rpctool` !

Protocoles sur Backdoor

AMOSSYS

RPCI

- Messages du système invité vers l'hyperviseur
- Informations sur le système invité, logs, drag n' drop, presse-papiers du guest, Unity...
- Exemple de commande : `info-set guestinfo.ip 192.168.122.123`
- Format des réponses `0/1`
- Utiliser l'outil `rpctool` !

TCLO

- Requêtes de l'hyperviseur vers le système invité
- Synchronisation de l'horloge, presse-papiers, commandes Unity...
- Exemple de commande : `Capabilities_Register`
- Format des réponses : `OK` ou `OK [DATA]`

Méthodologie RE (1) : repérage de symboles

AMOSSYS

RE : Par où commencer ?

- Nombreux binaires et DLLs, de taille importante
- On aimerait gagner du temps...
- Première étape : repérer des symboles
  - Exports des DLLs
  - Exploitation des *strings* des binaires

Méthodologie RE (1) : repérage de symboles

RE : Par où commencer ?

- Nombreux binaires et DLLs, de taille importante
- On aimerait gagner du temps...
- Première étape : repérer des symboles
  - Exports des DLLs
  - Exploitation des *strings* des binaires

```
lea     rdx, aVmxvmdb_setdev ; "VMXVmdb_SetDevPresent"
lea     rcx, aSDeviceNotFoun ; "%s: device not found!\n"
call    Debug
jmp     short loc_7FF7F15D63FF
```

Listing 2 – Exemple de log permettant de déterminer le nom de la fonction courante.

Méthodologie RE (2) : énumération de commandes RPCI

AMOSSYS

Repérage de commandes

- Inspection des *xrefs* à une commande connue
- Chaîne correspondante référencée dans une unique fonction commune à d'autres commandes, en charge d'enregistrer les *handlers* de commandes RPCI
- `RegisterRPCICommandHandler(id, vmdbKey, command, handler, a5)`

Méthodologie RE (2) : énumération de commandes RPCI

AMOSSYS

Repérage de commandes

- Inspection des *xrefs* à une commande connue
- Chaîne correspondante référencée dans une unique fonction commune à d'autres commandes, en charge d'enregistrer les *handlers* de commandes RPCI
- `RegisterRPCICommandHandler(id, vmdbKey, command, handler, a5)`

Utilisation

- Confirmation de l'existance avec `rpctool`
- Mise en avant de commandes inconnues/peu fréquentes dans les captures...
- ...et donc de nouveau éléments à *fuzz* :)
- Bémol : extrêmement chronophage de respécifier chaque commande !

AMOSSYS

Mise en évidence

- Utilisation du repérage de symboles
- Base de données relative à l'exécution de la machine virtuelle maintenue par le `vmx`
  - Informations sur le système invité
  - Gestion de la console (`MKSVmdb*`)
  - Callbacks (`VMXVmdbCb_RegisterCallbacks`)

VMDB

Mise en évidence

- Utilisation du repérage de symboles
- Base de données relative à l'exécution de la machine virtuelle maintenue par le `vmx`
  - Informations sur le système invité
  - Gestion de la console (`MKSVmdb*`)
  - Callbacks (`VMXVmdbCb_RegisterCallbacks`)

Propriétés

- Fonctionnement analogue à celui d'un système de fichiers...
  - Structure arborescente, chemins (`Vmdb_SetCurrentPath`), lien symbolique `..`
  - Exemple de chemin : `guest/caps/unityFeatures`
- ...tout en restant une base de données
  - Lecture (`Vmdb_Set*`) et écriture (`Vmdb_Get*`) de valeurs, notion de schéma

Communication inter-processus (1)

Architecture générale

- Découpage en processus
  - `vmware-vmx.exe` : hyperviseur, interaction avec `vmx86`
  - `vmware.exe` : interface graphique
- VMDB au coeur des communications entre ces processus

IPC via *named pipes*

- Le VMX expose des *named pipes*, ensuite utilisées par le processus `vmware.exe`
  - Mise à jour de données
  - Lancement d'évènements
- Sniffing des named pipes à l'aide de WinDbg : appels à `ReadFile`/`WriteFile`

Communication inter-processus (2)

```
6 TUPLESe /vm/#_VMX/vmx/1
1 21 021 unity/unityUpdate/#19/updateData/1 228
    ↪ bW92ZSA2NjQ1MiAzODIgNDgxIDQzMiA1MzkAAAA=1
1 22 220 1 10 1
1
```

Recherche de vulnérabilités sur le mode Unity

Fonctionnement du mode Unity (1)

AMOSSYS

Fonctionnement général

- Repose en grande majorité sur les mécanismes présentés
- Extensions de système invité : `vmtoolsd.exe`
  - Système de plugins (DLLs), différents niveaux de privilèges
- Basculement effectué à l'initiative du système hôte
- Échange de messages `unityUpdate` entre VM et hyperviseur, différents types

Fonctionnement du mode Unity (1)

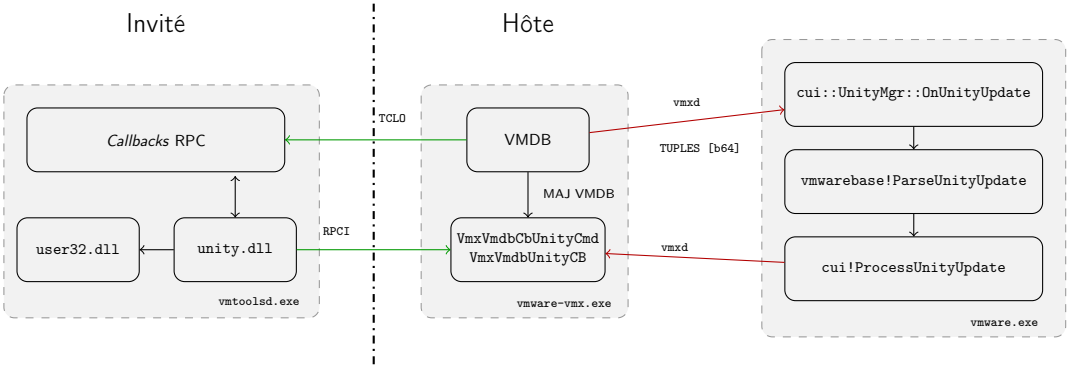
Fonctionnement général

- Repose en grande majorité sur les mécanismes présentés
- Extensions de système invité : `vmtoolsd.exe`
  - Système de plugins (DLLs), différents niveaux de privilèges
- Basculement effectué à l'initiative du système hôte
- Échange de messages `unityUpdate` entre VM et hyperviseur, différents types

Propagation des messages

- VM → hyperviseur : message RPCI → `unityUpdate` dans VMDB → IPC avec GUI → interaction `user32.dll`
- Hyperviseur → VM : message TCLO → prise en charge par `vmusr/unity.dll` → interaction `user32.dll`

Fonctionnement du mode Unity (2)



Problématique

- Mode Unity : communications complexes entre VM et VMM
- Surface d'attaque *a priori* intéressante
- Messages trop nombreux et complexes pour analyse manuelle

- Sous notre contrôle : messages RPCI et réponses TCLO
- Nécessité d'automatisation des interactions

Recherche de vulnérabilités sur le mode Unity

Solution retenue

Mutation et rejeu de captures RPCI

AMOSSYS

Méthodologie

- Méthode de capture présentée par ZDI
- Capture des messages RPCI dans une fonction du VMX avec WinDbg
- Rejeu avec un script dédié, dans l'ordre ou non

Avantages/défauts

+ Simplicité de mise en oeuvre
+ Génération automatique d'interactions **non cohérentes**
– *Code coverage* limité à moins de disposer une capture très importante
– Besoin de redéveloppement pour cibler d'autres mécanismes

MemITM (1)

AMOSSYS

Généralités

- *Fuzzer* en mémoire développé chez AMOSSYS
- Interception et modification **en mémoire** de messages d'un processus Windows depuis Python
- Disponible sur GitHub :)

Utilisation

- Génération d'une configuration via IDAPython
- Deux fonctions exposées à l'utilisateur
  - `logger` : sauvegarde des messages, envoi via HTTP...
  - `fuzzer` : *bitflip*, substitution de valeurs...
- Utilisation des fonctions et bibliothèques Python

MemITM (2)

Placement du *hook* en mémoire

- Nécessité de savoir où se placer en mémoire
- Fuzzing d'une partie ou de toutes les commandes : en fonction de l'adresse du *hook*
- Dans notre cas : placement dans les DLLs de `vmtoolsd.exe` (pas de *bypass* de *checks*)

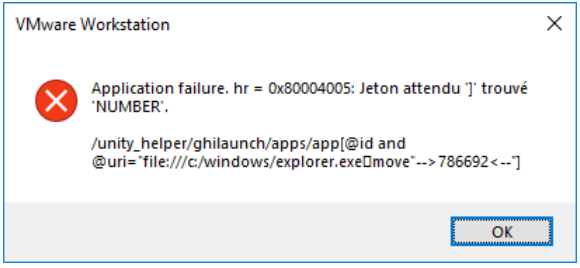
Avantages/défauts

+ Précision dans le ciblage des messages à muter
+ Pas de besoin de redéveloppement, il suffit de modifier le *hook*
– Création des messages non automatique
– Impossible de modifier la taille du buffer...

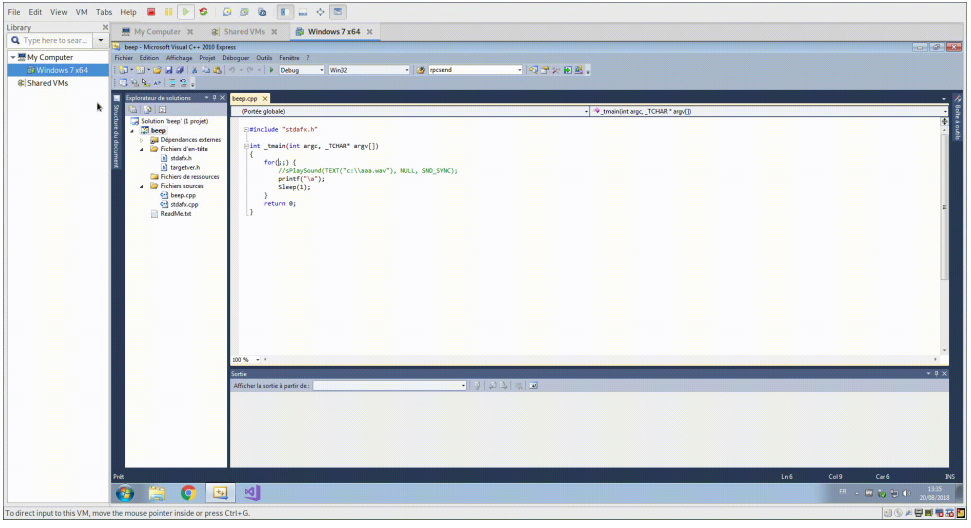
Recherche de vulnérabilités sur le mode Unity

Résultats

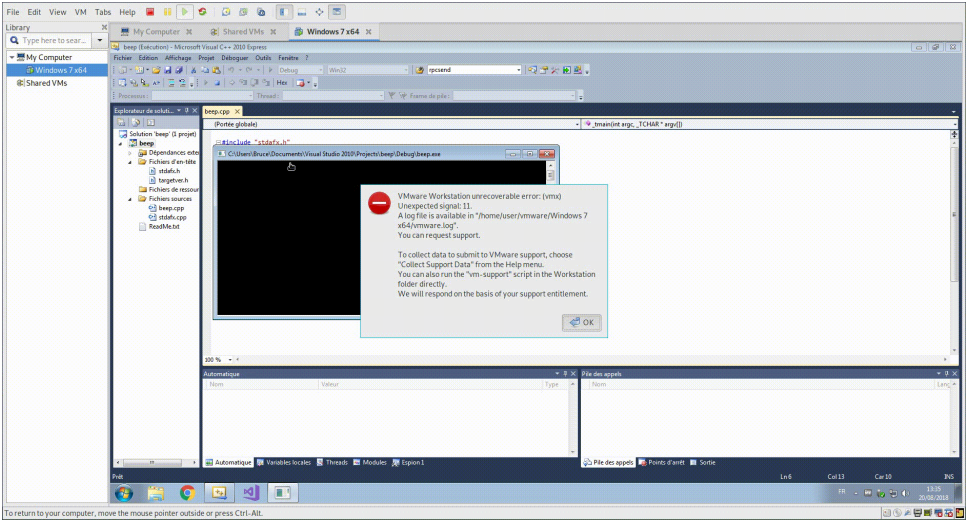
Ciblage de la GUI ?



Crash en quelques `printf` (1)



Crash en quelques `printf` (1)



Crash en quelques `printf` (2)

AMOSSYS

Au secours, `printf` a cassé mon hyperviseur !

- Vulnérabilité découverte « par hasard » : affichage des `0x07` lors du rejeu de messages RPCI
- Bug dans le transfert de l'audio de la VM vers l'hôte, **non privilégié**
- Spécifique à la version Linux de VMware Workstation
- Signalée à VMware et corrigée (CVE-2019-5525)

Crash en quelques `printf` (2)

AMOSSYS

Au secours, `printf` a cassé mon hyperviseur !

- Vulnérabilité découverte « par hasard » : affichage des `0x07` lors du rejeu de messages RPCI
- Bug dans le transfert de l'audio de la VM vers l'hôte, **non privilégié**
- Spécifique à la version Linux de VMware Workstation
- Signalée à VMware et corrigée (CVE-2019-5525)

Exploitabilité ?

- Reproductibilité : sur divers systèmes hôtes et invités
- Différents cas :
  - Écriture à une adresse invalide
  - Appel d'une fonction après déréférence d'un pointeur : besoin d'un *info leak*
- *Heap spraying* à l'aide de messages RPCI

Cas 1 : écriture à une adresse invalide

```
-------[registers]-------
$rax: 0x0
$rbx: 0x8b703
$rcx: 0x0
$rdx: 0x0
$rsp: 0x7ffdbdf6c908 → [...] → 0x0000000000424242 ("BBB"?)
$rbp: 0x0
$rsi: 0x7f95ac493410 → 0x00007f95ac3d9460 → "BBBBBBBBBBBBBBBB[...]"
[...]
$r10: 0x0
[...]
$r13: 0x7f95ac30f310 → 0x00007f95ac4462c0 → 0x0042424242424242 ("BBBBBBB"?)
[...]
---------[code]----------
0x7f95bb8c0147     nop WORD PTR [rax+rax*1+0x0]
0x7f95bb8c0150     add r14d, 0x1
0x7f95bb8c0154     cmp DWORD PTR [rdi+0x10], r14d
0x7f95bb8c0158     mov WORD PTR [r10], r11w ← $rip
0x7f95bb8c015c     ja 0x7f95bb8c001f
0x7f95bb8c0162     pop rbx
0x7f95bb8c0163     pop rbp
```

Cas 2 : *VM escape* potentielle

```
-------[registers]-------
$rax: 0x4242424242424242 ("BBBBBBBB"?)
$rbx: 0x7fe7b81f4800 → 0x0000000000000000
$rcx: 0x6f
[...]
$rdi: 0x4242424242424242 ("BBBBBBBB"?)
$rip: 0x7fe7c5777681 → call QWORD PTR [rax+0x38]
$r8: 0x0
[...]
---------[code]----------
0x7fe7c577766e     add BYTE PTR [rbp+0x484d75f6], al
0x7fe7c5777674     mov eax, DWORD PTR [rbx+0x198]
0x7fe7c577767a     mov rdi, QWORD PTR [rbx+0x1a8]
0x7fe7c5777681     call QWORD PTR [rax+0x38] ← $rip
0x7fe7c5777684     mov edx, DWORD PTR [rbx+0x1cc]
0x7fe7c577768a     mov ebp, eax
0x7fe7c577768c     test edx, edx
0x7fe7c577768e     jne 0x7fe7c57776a0
```

Conclusions et perspectives

Conclusions

- Mode Unity : surface d'attaque *a priori* intéressante, intérêt final limité
  - Passage initié depuis le système hôte
  - Pas de crash du VMX relatif au mode Unity (*fuzzing* « trivial »...)
  - *Fuzzing* en mémoire : perte du suivi des fenêtres
- Mise en avant de nouvelles surfaces d'attaques potentielles : audio et GUI

Perspectives

- Fuzzing audio de l'hyperviseur VMware
- Recherche de vulnérabilités relative à l'interface graphique (Workstation et ESXi)

Questions ?