

CrashOS

Recherche de vulnérabilités système dans les hyperviseurs

SSTIC 2017

Anaïs GANTET - Airbus Group Innovations

8 juin 2017

Plan

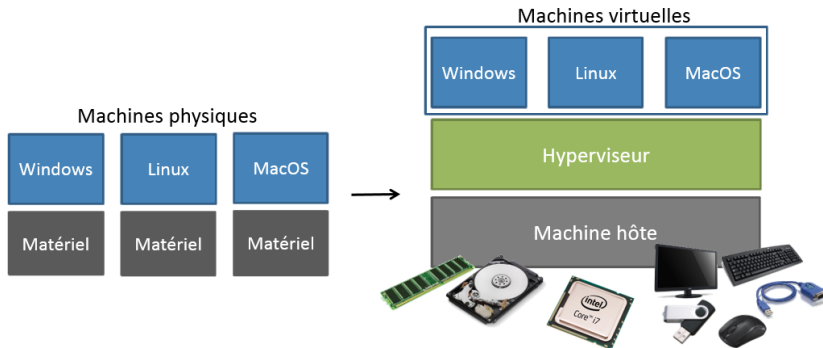
- 1 CrashOS : motivations
- 2 CrashOS : présentation
- 3 CrashOS : recherche de vulnérabilités et résultats

Plan

- 1 CrashOS : motivations
- 2 CrashOS : présentation
- 3 CrashOS : recherche de vulnérabilités et résultats

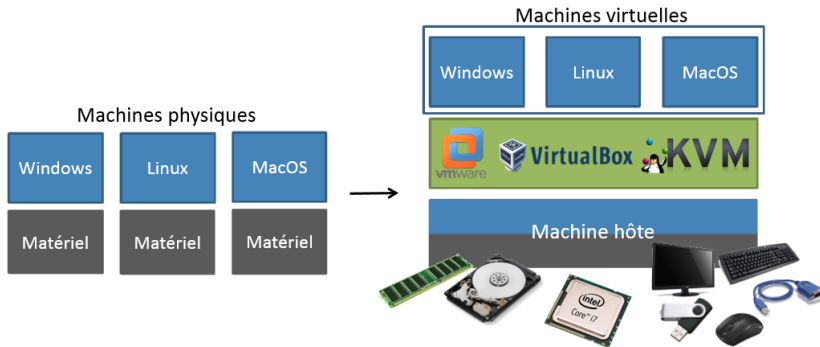
La virtualisation en bref

But : faire fonctionner plusieurs systèmes d'exploitation (OS) sur une même machine physique



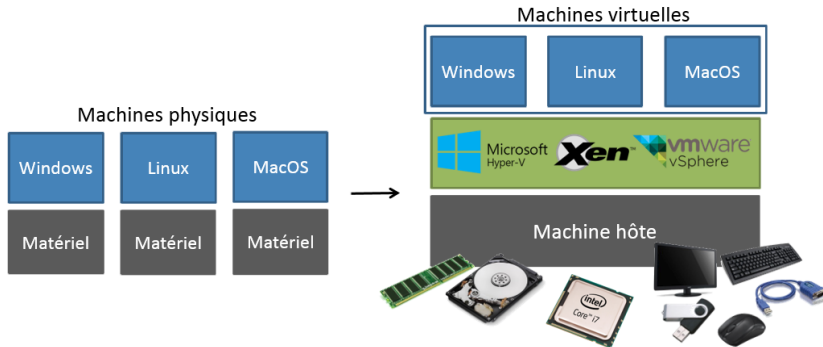
La virtualisation en bref

But : faire fonctionner plusieurs systèmes d'exploitation (OS) sur une même machine physique



La virtualisation en bref

But : faire fonctionner plusieurs systèmes d'exploitation (OS) sur une même machine physique



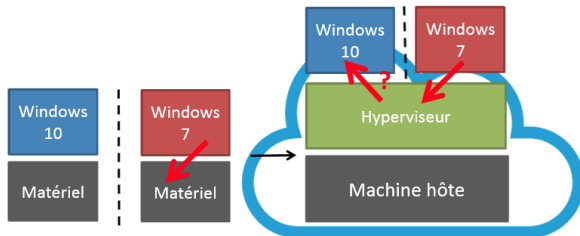
Rôle d'un hyperviseur

Fournir un environnement équivalent

- Virtualisation du processeur (Intel)
 - Laisser s'exécuter les instructions non sensibles
 - Intercepter et virtualiser les instructions sensibles
 - Virtualisation de l'accès aux ressources mémoire
 - Virtualisation des périphériques
-
- Code complexe
 - Intervention niveau système

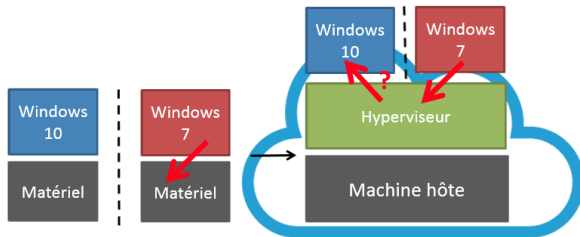
Risques de sécurité

- Sans virtualisation : isolation physique
- Avec virtualisation : isolation logicielle



Risques de sécurité

- Sans virtualisation : isolation physique
- Avec virtualisation : isolation logicielle



Problématique

Les hyperviseurs actuels sont-ils robustes ?

Notre approche

Recherche par analyse comportementale

- Construire une VM à configuration système atypique
- Lancer des tests mettant en jeu un traitement de l'hyperviseur
- Observer le comportement des hyperviseurs

Possibilités de mise en œuvre

- Windows, Linux : contrôle partiel de la communication avec le matériel
- Simple Operating System, OSv, etc. : peu adaptés à la recherche de failles
- VESPA : outil se limitant à la recherche de failles dans les périphériques

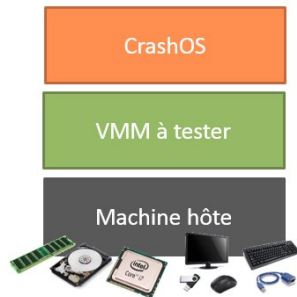
Solution retenue : proposer un outil adapté aux tests des hyperviseurs

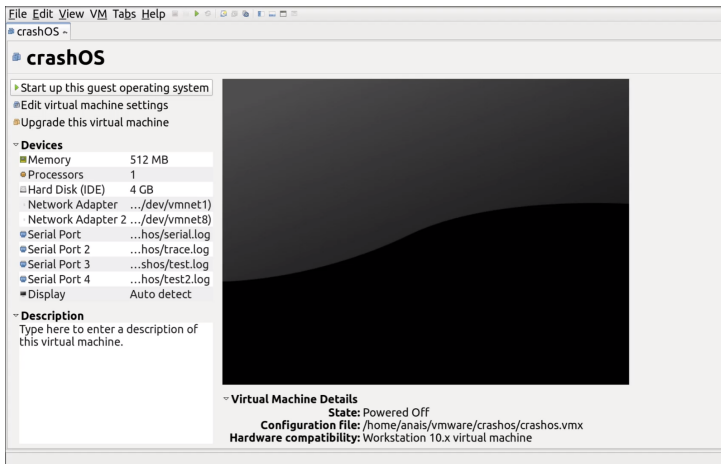
Plan

- 1 CrashOS : motivations
- 2 CrashOS : présentation**
- 3 CrashOS : recherche de vulnérabilités et résultats

CrashOS : Le projet

- OS minimaliste, open-source (licence GPLv2)
- Langage C et assembleur (Intel)
- 2 mois de développement (API)
- Lancé sur Ramooflax, VMware, Xen (HVM)





CrashOS : Core (API)

Fonctionnalités système

- Accès à la mémoire physique
- Mécanisme de protection mémoire
- Gestion des interruptions
- Communication avec les périphériques
- Support de la paravirtualisation

Exemple :

```
#define enable_paging()  
asm volatile (  
    "mov %%cr0, %%eax          \n"  
    "or $0x80000000, %%eax     \n"  
    "mov %%eax, %%cr0"::"eax" )
```

CrashOS : Core (API)

Fonctionnalités système

- Accès à la mémoire physique
- Mécanisme de protection mémoire
- Gestion des interruptions
- Communication avec les périphériques
- Support de la paravirtualisation

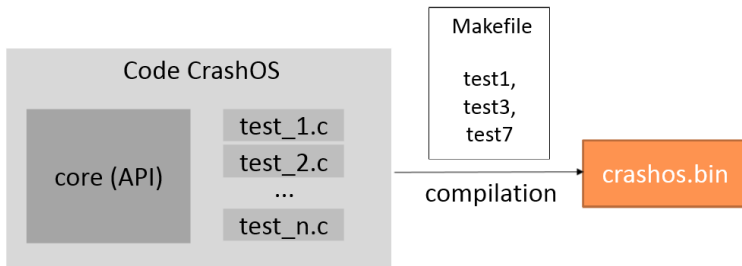
Exemple :

```
#define enable_paging()  
asm volatile (  
    "mov %%cr0, %%eax          \n"  
    "or $0x80000000, %%eax     \n"  
    "mov %%eax, %%cr0"::"eax" )
```

Fonctionnalités de lancement des tests

- Macro DECLARE_TEST(test_x) sur chaque test
- Lancement en série des tests répertoriés
- Affichage de messages à l'écran ou sur le port série

CrashOS : Principe de fonctionnement



CrashOS : Format d'un test

- Fonction d'initialisation
 - Sauvegarde de l'état courant
 - Définition d'un contexte particulier
- Fonction du test en lui-même
 - Exécution de l'action à déclencher
 - Affichage de logs adaptés au test
- Fonction de restauration
 - Réinitialiser l'état sauvegardé

```
test_t test_x = {  
    .name = "test name",  
    .desc = "test description",  
    .init = init_test_x,  
    .test = test_test_x,  
    .fini = restore_test_x  
};  
DECLARE_TEST(test_x)
```

Plan

- 1 CrashOS : motivations
- 2 CrashOS : présentation
- 3 CrashOS : recherche de vulnérabilités et résultats**

Démarche d'élaboration d'attaque

Élaboration de tests pertinents

- Compréhension poussée des rouages du processeur Intel
- Bonne connaissance du rôle des hyperviseurs

Points critiques identifiés (CVE)

- Désassemblage des instructions
- Émulation des instructions sensibles
- Virtualisation ou émulation des périphériques
- etc.

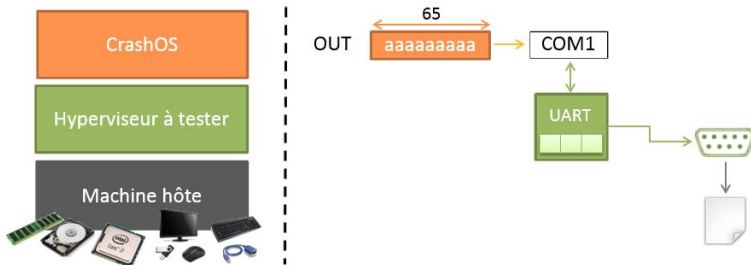


Quelques cas de tests dans CrashOS

| | I/O | privileges | segmentation | pagination | modes | nested virt. | instructions | données | VMware | Ramooflax | Xen(HVM) |
|---|-----|------------|--------------|------------|-------|--------------|--------------|---------|--------|-----------|----------|
| REP OUTS COM3 - DS sur 2 pages non contiguës | x | | x | x | | | | x | | | |
| REP OUTS COM3 - Ring 3 avec DS sur une page ring 0 | x | x | | x | | | | x | | | |
| REP OUTS COM3 - Ring 3 avec problème de droits | x | x | | | | | | x | | | |
| REP OUTS COM1 - @data > limite | x | | x | | | | | x | | | |
| REP OUTS COM1 - DS en expand-down | x | | x | | | | | x | | | |
| REP OUTS COM1 - taille > 130 octets | x | | | | | | | x | | | |
| REP OUTS COM1 - Instruction mode 16 | x | | | | x | | | x | | | |
| OUT 0xb2 (#SMI) - fuzzing avec valeurs random | x | | | | x | | | | | | |
| Instructions privilégiées en ring 3 | | x | x | | | | | x | | | |
| Instructions sensibles non-priviliégées - violation de droits | | x | | x | | | | x | | | |
| Instruction placée sur 2 pages | | x | | x | | | | x | | | |
| Toute valeur sur n'importe quel port d'I/O | x | | | | | | | x | | | |
| LJMP avec opérande en MMIO et mauvais droits | x | x | x | | | | | x | | | |
| Task switch avec opérande en MMIO | x | x | x | | | | | x | | | |

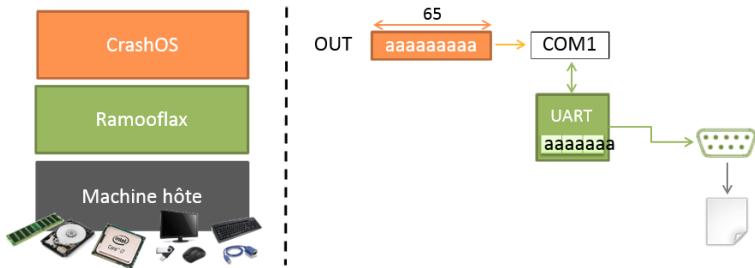
Un premier exemple : écriture d'un grand buffer sur le port série COM 1

```
// REP OUTS instruction
asm volatile (
    "mov $0x3f8,    %%edx \n" /* serial port COM 1 */
    "mov $0x400000, %%esi \n" /* buffer address */
    "mov $65,      %%ecx \n" /* number of byte */
    "rep outsb     \n" :::);
```



Un premier exemple : écriture d'un grand buffer sur le port série COM 1

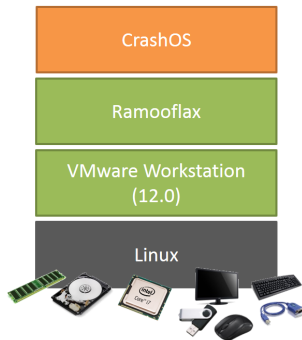
```
// REP OUTS instruction
asm volatile (
    "mov $0x3f8,    %%edx \n" /* serial port COM 1 */
    "mov $0x400000, %%esi \n" /* buffer address */
    "mov $65,      %%ecx \n" /* number of byte */
    "rep outsb     \n" :::);
```



Buffer overflow et corruption mémoire de la structure info de Ramooflax



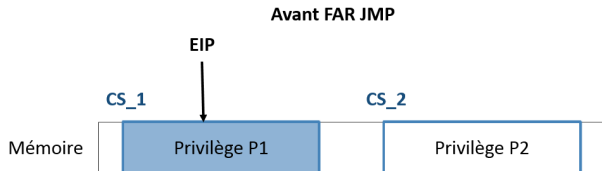
Un premier exemple : écriture d'un grand buffer sur le port série COM 1



Analyse du bug : *Monitor panic* de Vmware lorsque @PTE mal configurée

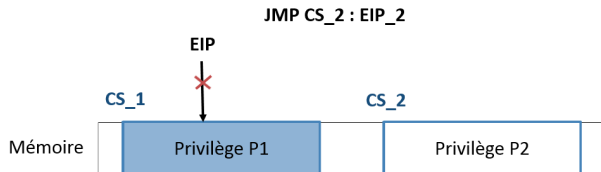
Un deuxième exemple : changement de privilège (FAR JMP)

Rappels : fonctionnement du FAR JMP



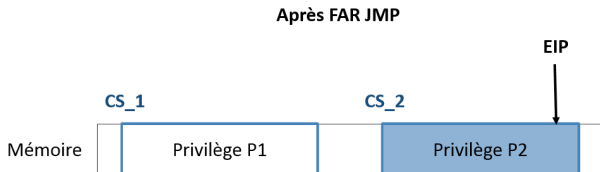
Un deuxième exemple : changement de privilège (FAR JMP)

Rappels : fonctionnement du FAR JMP



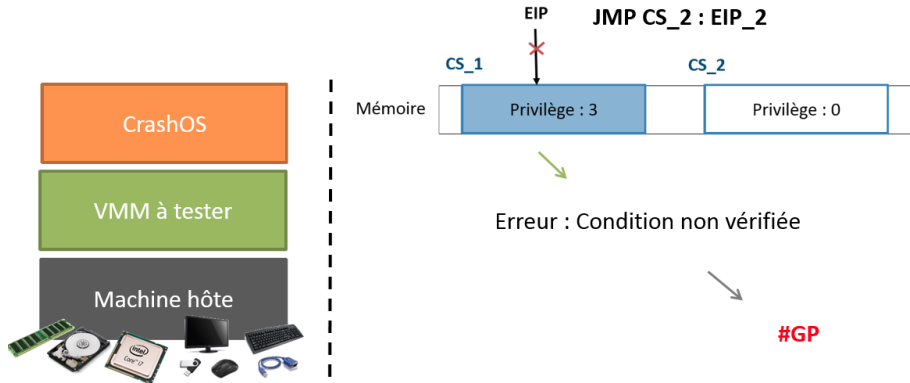
Un deuxième exemple : changement de privilège (FAR JMP)

Rappels : fonctionnement du FAR JMP



- Remarque : transfert réussi seulement si les paramètres respectent plus d'une vingtaine de conditions
- But de l'attaque : s'assurer que les hyperviseurs vérifient bien chacune des conditions (inspiré de CVE-2014-8595 - Xen HVM)

Un deuxième exemple : changement de privilège (FAR JMP)



Un deuxième exemple : changement de privilège (FAR JMP)

Cas de Xen (HVM) :

```
root@debian:~# xl list
```

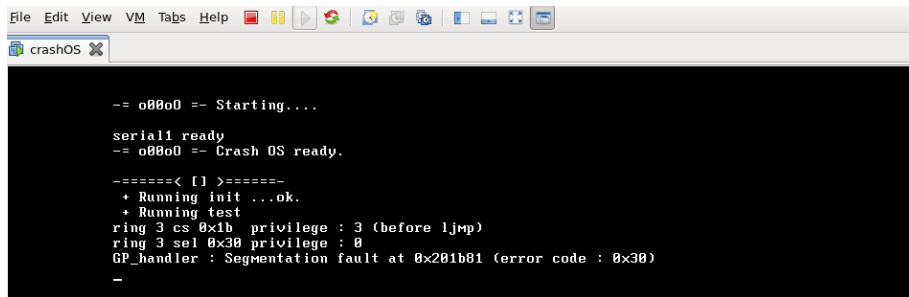
| Name | ID | Mem | VCPUs | State | Time(s) |
|----------|----|-----|-------|--------|---------|
| Domain-0 | 0 | 395 | 1 | r----- | 220.9 |
| crashos | 28 | 507 | 1 | ---sc- | 5.7 |

```
root@debian:~# xl dmesg_
```

```
(XEN) domain_crash called from vmx.c:2274
(XEN) Domain 28 (vcpu#0) crashed on cpu#0:
(XEN) ----[ Xen-4.2.4-pre x86_32p debug=n Not tainted ]----
(XEN) CPU: 0
(XEN) EIP: 0030:[<00201b95>] EIP_2
(XEN) EFLAGS: 00000006 CONTEXT: hvm guest
(XEN) eax: 000b8000 ebx: 0002dae0 ecx: 000b8000 edx: 000b8000
(XEN) esi: 00054769 edi: 0005476a ebp: 0020721c esp: 0020721c CS_2
(XEN) cr0: 00000011 cr4: 00000000 cr3: 00800000 cr2: 00000000
(XEN) ds: 0023 es: 0000 fs: 0000 gs: 0000 ss: 0023 cs: 0030
```

Un deuxième exemple : changement de privilège (FAR JMP)

Cas de VMware :

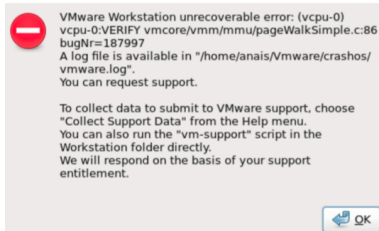


```
-- o00o0 -- Starting....  
  
serial1 ready  
-- o00o0 -- Crash OS ready.  
  
=====< [] >=====  
+ Running init ...ok.  
+ Running test  
ring 3 cs 0x1b privilege : 3 (before ljmp)  
ring 3 sel 0x30 privilege : 0  
GP_handler : Segmentation fault at 0x201b81 (error code : 0x30)  
-
```

Divers résultats

Crashes de la VM

- VMware et pagination
 - *Monitor panic* VERIFY
- VMware et périphériques
 - *Monitor panic* VERIFY
 - *Monitor panic* NOT_IMPLEMENTED
 - *Monitor panic* NOT_REACHED
- VMware et *nested virtualization*
 - *Monitor panic* EPT Misconfiguration
- KVM : *KVM internal error. Suberror : 3*



Autres

- Ramooflax : Contrôle de la VM (buffer UART)
- Ramooflax : Mauvaise résolution d'adresse (V8086)

Conclusion et perspectives

CrashOS aujourd'hui

- Opensource
- OS minimaliste configurable
- Rédaction simple d'attaques système sur les hyperviseurs
- Premiers résultats sous VMware et Ramooflax

Perspectives de l'outil

- De nouveaux tests à élaborer
- De nouvelles fonctionnalités à implémenter (64 bit, *nested virtualization*, etc.)
- D'autres hyperviseurs à tester (Xen PV, KVM, Virtualbox, etc.)

Merci pour votre attention

Des questions ?

`https://github.com/airbus-seclab/crashos`

`anais.gantet@airbus.com`