Sécurité des systèmes d'exploitation

Protection du noyau et de l'espace utilisateur

Plan

- Concepts communs
  - Protection de la mémoire
  - Protection contre les *buffer overflow*
- Protection du noyau Linux
- Linux Security Modules
  - AppArmor
- Protection Windows
  - PatchGuard et KMCI
  - AppLocker

Protection de la mémoire

- La gestion de la mémoire est l'élément central de la sécurité d'un système pour
  - La protection du noyau vis-à-vis des applications utilisateur
  - Le cloisonnement entre les applications
- Rappel : dans le processeur, la MMU (*Memory Management Unit*) est en charge de l'accès à la RAM et implémente les mécanismes de
  - Segmentation : transformation d'une adresse logique en une adresse linéaire
  - Pagination : traduction de cette adresse linéaire en une adresse physique

Protection de la mémoire

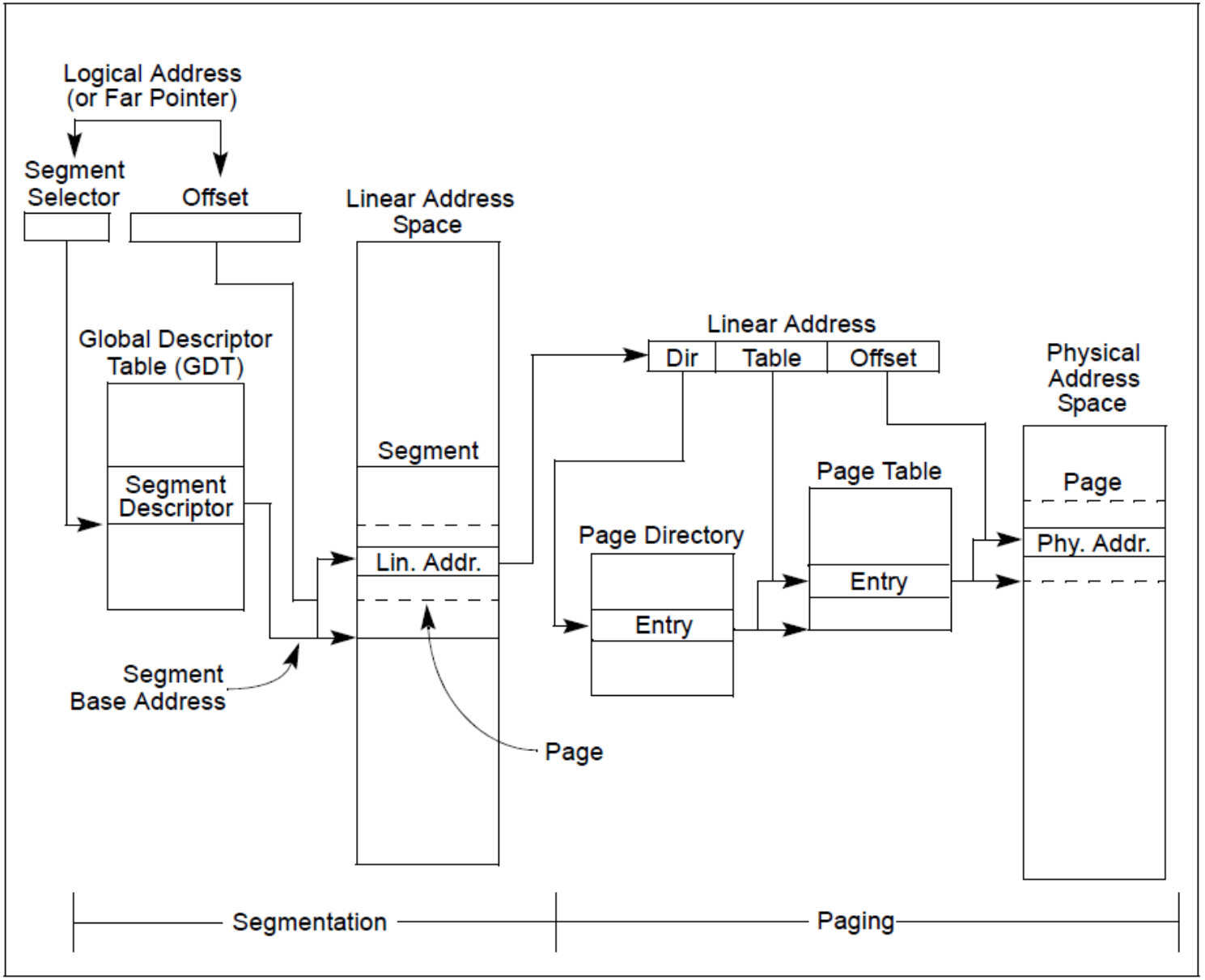


Figure 3-1. Segmentation and Paging

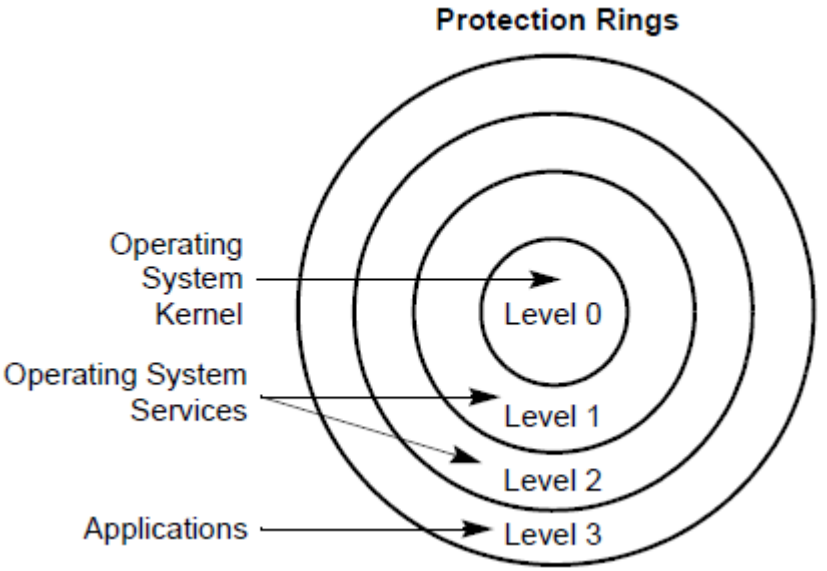
Source : Intel

Protection de la mémoire

- Modes d'exécution d'un processeur de type x86
  - Réel : mode de démarrage (BIOS, bootloader), adressage 16 bits, un seul niveau de privilège
  - Protégé : mode d'exécution nominal 32 bits
  - Long : mode d'exécution nominal 64 bits
  - *System Management* (SMM) : mode de maintenance, accès à toute la mémoire
    - bascule dans ce mode sur réception de SMI (*System management Interrupt*) liées généralement à des événements matériels (par ex. gestion de l'énergie)
    - Exécution d'une routine de traitement écrite par le fabricant de la carte mère
    - Utilise une partie réservée de la RAM : la SMRAM

Protection de la mémoire

- Niveau de privilèges des processeurs x86
  - 4 niveaux de privilèges, aussi appelés anneaux (« rings ») de protection, numérotés de 0 (plus haut niveau de privilèges) à 3 (plus bas niveau de privilèges), disponibles dans :
    - Mode protégé (32 bits)
    - Mode long (64 bits)
  - En pratique, on retrouve surtout les niveaux 0 et 3
  - Certaines instructions, dites privilégiées, nécessitent le niveau de privilèges 0
    - Par ex. HLT, INVD, INPLG, ...



Source : Intel

Protection de la mémoire

- Niveau de privilèges des processeurs x86
  - Le champ CPL (Current Privilege Level) du registre de segment CS indique le niveau de privilège du code en cours d'exécution.
  - Lorsque du code exécuté en espace utilisateur (i.e. CPL = 3) a besoin d'accéder à des ressources contrôlées par le noyau, il réalise un appel système (*syscall*) pour exécuter du code noyau (i.e. CPL = 0)
  - Le changement de niveau de privilège se fait :
    - soit en déclenchant une interruption logicielle (instruction *INT*) ;
    - soit par l'instruction de retour de procédure (*RET*) ;
    - soit à l'aide d'instructions spécifiques (*SYSENTER* / *SYSEXIT* ou *SYSCALL* / *SYSRET*).

Protection de la mémoire

- Niveau de privilèges des processeurs x86
  - En cas d'interruptions matérielles ou d'exceptions, un changement de niveau peut avoir lieu
    - par ex. traitement par une routine du noyau
  - De façon simplifiée, un numéro d'interruption (ou d'exception) sert d'index dans la table IDT (*Interrupt Descriptor Table*) pour identifier la routine à utiliser, ainsi que le niveau CPL pour exécuter cette routine
    - Table chargée à partir de l'instruction LIDT
      - Extrait du manuel Intel : *The LIDT instruction loads the IDTR register with the base address and limit held in a memory operand. This instruction can be executed only when the CPL is 0. It normally is used by the initialization code of an operating system when creating an IDT. An operating system also may use it to change from one IDT to another.*

Protection de la mémoire

- Segmentation dans les processeurs x86 (32 bits)
  - Un sélecteur de segment est un identifiant unique d'un segment dans la GDT (*Global Descriptor Table*) ou une LDT (*Local Descriptor Table*), il contient un champ RPL (*Requested Privilege Level*)
    - Refus d'accès si le CPL > RPL
    - Abaissement du CPL à la valeur du RPL si CPL < RPL
  - Un descripteur de segment contient un champ DPL (*Descriptor Privilege Level*) détermine le niveau de privilège d'un segment vis-à-vis du CPL
    - En principe, il faut que CPL ≤ RPL
    - En pratique, la logique de contrôle d'accès est assez complexe et varie, notamment, en fonction du type de segment (code, données, pile)

Protection de la mémoire

- Segmentation dans les processeurs x86 (32 bits)
  - Un contrôle est effectué sur la limite d'un segment pour empêcher d'accéder en dehors de ce segment
  - Activée lors du passage dans le mode protégé
- Segmentation dans les processeurs x86 (64 bits)
  - Pour le code 32 bits (mode comptabilité), pas de changement
  - Pour le code 64 bits, la segmentation est en partie désactivée
    - Adresse linéaire = adresse effective
    - Pas de contrôle sur la limite d'un segment

Protection de la mémoire

- Pagination dans les processeurs x86
  - Par défaut, une entrée de la table des pages contient deux drapeaux :
    - le drapeau U/S (*User*/*Supervisor*) qui détermine le type de la page
    - le drapeau R/W (*Read*/*Write*) qui détermine le type d'accès autorisé à une page (lecture seule ou lecture-écriture)
      - Avec ce modèle, une page est toujours exécutable
  - L'extension PAE (Physical Address Extension) et le mode 64 bit ajoute le drapeau XD (eXecute Disable) qui permet d'empêcher l'exécution d'une page mémoire.

Protection de la mémoire

- Pagination dans les processeurs x86
  - Par défaut, lors de l'accès à une adresse linéaire, 2 niveaux sont considérés à partir du CPL :
    - le mode *supervisor* correspond à CPL < 3
    - le mode *user* correspond à CPL = 3
  - Sauf pour l'accès à des structures de données comme, entre autres, les tables GDT, LDT ou IDT
    - Accès « implicite » en mode *supervisor*, indépendamment du CPL

Protection de la mémoire

- Pagination dans les processeurs x86
  - logique d'accès assez complexe, quelques éléments à retenir
    - Du code en mode *user* ne devrait pas :
      - Accéder en lecture ou écriture à une adresse *supervisor*
      - Récupérer des instructions à une adresse *supervisor*
    - Du code en mode *supervisor* peut accéder en lecture à une adresse *supervisor*
    - Les bits de plusieurs registres de contrôle sont utilisés pour les autres cas, notamment
      - CR4.SMEP (*supervisor-mode execution prevention*) : permet de d'empêcher des pages d'être exécutées en mode *supervisor*
        - Évite l'exécution de code en espace utilisateur par le noyau
      - CR4.SMAP (*supervisor-mode access prevention*) : permet de d'empêcher des pages d'être accédées en mode *supervisor*
        - Sauf si le bit EFLAGS.AC = 1

Protection de la mémoire

- Le noyau Linux et Windows utilisent les niveaux de privilèges des processeurs x86 suivants :
  - 0 pour l'espace d'adressage du noyau ;
  - 3 pour l'espace d'adressage des processus utilisateurs.
- Le noyau Linux et Windows utilisent la segmentation et la pagination avec une architecture x86
  - La segmentation est généralement sous-utilisée, souvent pour des questions de portabilité vers d'autres architectures processeur
  - En 64 bits, la segmentation est peu utilisée puisque presque désactivée

Protection contre les *buffer overflow*

- Rappels
  - Les programmes disposent de 2 « zones » mémoires : le tas et la pile
  - Un *buffer overflow* consiste à écraser des données dans l'une de ces zones en profitant d'une erreur logique
  - Exemples d'exploitations
    - Pile : modification de l'adresse de retour (autre partie du programme, shellcode placé sur ou en dehors de la pile, libc, ...)
    - Tas : pointeurs de fonctions, gestionnaire d'exception

Protection contre les *buffer overflow*

- Il existe d'autres classes d'attaques qui permettent de modifier le flot d'exécution d'un programme
  - *Untrusted Pointer Dereference*, Expired Pointer *Dereference*, ...
- Les protections disponibles se répartissent entre le compilateur (canaris, Control Flow Integrity, …) et le système d'exploitation
  - Souvent, il faut combiner les 2 pour une meilleure efficacité
    - Par ex., le chargeur d'exécutable a besoin de savoir qu'une zone est non-exécutable pour positionner correctement la protection adaptée sur la page mémoire

Protection contre les *buffer overflow*

- ASLR (*Address Space Layout Randomization*)
  - Modification du mécanisme d'allocation mémoire qui conduit à placer différents objets à des adresses aléatoires plutôt qu'à des adresses fixes
    - Vise à lutter contre les techniques du type *return-to-libc*
    - Son efficacité dépend de l'espace des adresses possibles, c.a.d. du nombre de bits (qualifié d'entropie) pouvant varier dans l'adresse
- W^X (*Write xor eXec*)
  - Principe qui stipule qu'une zone de mémoire ne doit pas être à la fois modifiable et exécutable
    - Une zone qui contient du code ne doit pas être modifiable
    - Une zone qui contient des données (par ex., le tas ou la pile) ne doit pas pouvoir être exécutée

Protection contre les *buffer overflow*

- Prise en compte de l'ASLR sous Linux (depuis la version 2.6.12) pour les processus
  - Activé si /proc/sys/kernel/randomize_va_space >= 1
  - S'applique, entre autres, à la pile, au tas, au code du programme et aux bibliothèques
  - Nécessite que les binaires soient compilés et liés avec l'option PIE (Position Independent Executable) pour être complètement efficace
- Prise en compte de kASLR depuis le noyau 3.14 pour les processeurs x86 (32 et 64 bits)
  - Paramètre CONFIG_RANDOMIZE_BASE
    - Désactivable avec l'option noaslr dans la ligne de commande du noyau

Protection contre les *buffer overflow*

- Certaines régions de la mémoire (pile, tas) sont rendues non-exécutables
  - Utilise le bit XD (ou NX selon le constructeur) des entrées de table de page
  - Se vérifie avec `grep NX /var/log/dmesg`

```
[    0.000000] NX (Execute Disable) protection: active
```

Protection contre les *buffer overflow*

- ASLR sous Windows
  - Introduit dans Windows Vista
  - S'applique à la pile, au tas, au PEB, au TEB, aux programmes et aux bibliothèques
  - Nécessite une prise en compte par les binaires (/DYNAMICBASE)
  - L'entropie théorique varie en fonction des versions de Windows et de l'architecture processeur (32 ou 64 bits)
    - Par ex., elle est de 8 bits pour Windows 7 x64 et de 17 bits pour Windows 8 x64
  - kASLR depuis Windows 8 (noyau NT et chargement des pilotes)

Protection contre les *buffer overflow*

- Data Execution Prevention (DEP)
  - Marquage de certaines pages comme non-exécutables (utilisation du bit NX)
  - 4 modes de fonctionnement
    - « Opt-In » : activé pour les processus qui accepte explicitement le DEP (/NXCOMPAT). Par défaut pour les systèmes Windows client (Vista, 7, 8, …) ;
    - « Opt-Out »: activé pour tous les processus sauf ceux qui refusent explicitement le DEP. Par défaut pour les systèmes Windows serveur (2008, 2012, …) ;
    - « Always On » : activé pour tous les processus ;
    - « Always Off » : désactivé pour tous les processus.
- depuis Windows 8, l'espace noyau intègre également une protection de type W^X

Protection du noyau Linux

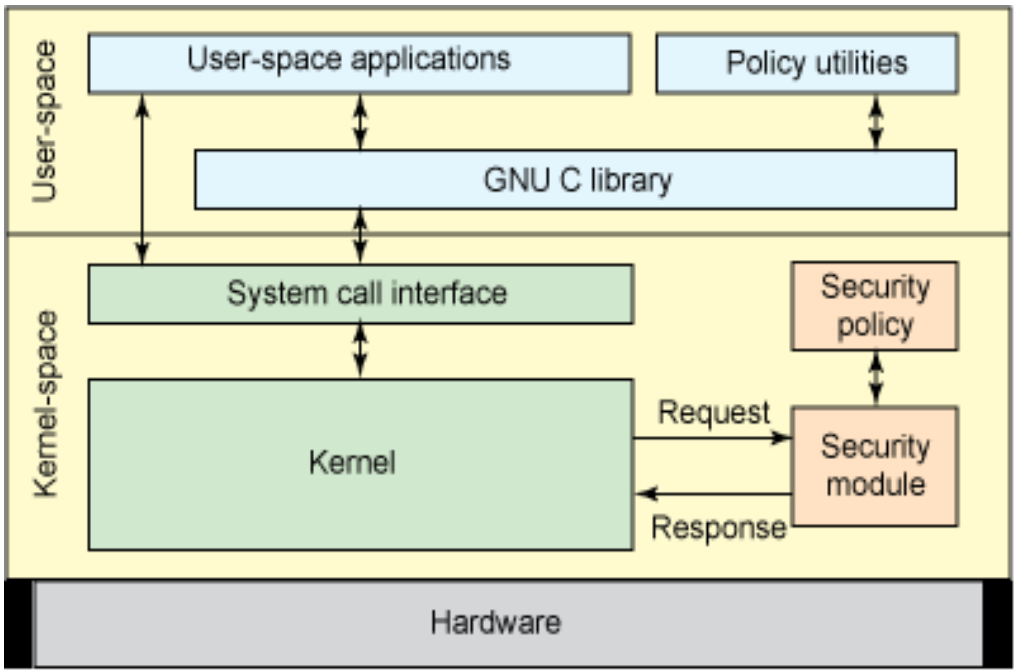
- Signature cryptographique des modules noyau
  - Disponible depuis le noyau Linux 3.7
  - Les modules sont signés lors de la construction du noyau
    - Utilisation de certificats X509 pour les clés publiques RSA
  - La signature est vérifiée par le noyau lors du chargement du module
  - Désactive le chargement de modules non-signés ou signés avec une clé invalide
    - Complique le chargement de modules malveillants (par ex., un *rootkit*) dans le noyau
    - Attention, la vérification est permissive si le paramètre CONFIG_MODULE_SIG_FORCE n'est activé

Protection du noyau Linux

- D'autres mécanismes existent, notamment avec le patch GrSecurity (payant)
  - Système RBAC
  - PaX : protection contre certains type d'exploits
  - Restriction sur chroot() et divers durcissements
  - TPE (*Trust Path Execution*) : liste blanche de fichiers pouvant être exécutés
- KSPP (*Kernel Self Protection Project*)
  - Projet visant à intégrer de nouvelles protections en standard dans le noyau sans passer par un patch comme GrSecurity

Linux Security Modules

- Rappel : le noyau Linux propose une interface pour intégrer des modules noyaux qui apportent des fonctions de contrôle d'accès
  - Ces modules sont appelés des Linux Security Modules
    - Liste des modules actifs : /sys/kernel/security/lsm



Source : IBM

Linux Security Modules

- En plus des modules MAC (SELinux, SMACK, Tomoyo, AppArmor), il existe également des LSM assurant d'autres fonctions de sécurité :
  - IMA/EVM (*Integrity Measurement Architecture / Extended Verification Module*): détection de changements accidentels ou malveillants des fichiers protégés, qu'ils soient locaux ou distants
    - Évalue (*appraise*) une mesure de fichier vis-à-vis d'une valeur de référence stockée dans un attribut étendu (security.ima) signé du système de fichiers
      - Peut bloquer le chargement d'un fichier modifié
    - Repose sur un TPM, notamment pour protéger (via la SRK) les clés utilisées pour la signature des attributs ou les clés utilisées pour les calculs des HMAC des attributs security.*

Linux Security Modules

- (suite)
  - Yama : contrôle fin de l'étendu de l'appel ptrace()
    - Paramètre /proc/sys/kernel/yama/ptrace_scope
      - 0 : permissions ptrace classique
      - 1 : ptrace restreint, attachement limité par défaut d'un processus parent à un processus enfant (comportement modifiable avec prctl() pour déclarer le PID d'un *debugger*)
      - 2 : attachement limité à un administrateur, i.e. un processus avec la capacité CAP_SYS_PTRACE
      - 3 : pas d'attachement
  - LoadPin : s'assure que tous fichiers chargés par le noyau (modules, *firmware*, ...) proviennent du même système de fichiers
    - Utilisé avec dm-verity dans certaines configurations Android

AppArmor

- Système MAC implémenté au moyen d'un module de sécurité pour le noyau Linux (LSM) et destiné à confiner les programmes à un ensemble limité de ressources.
  - Le contrôle d'accès réalisé s'applique uniquement sur les programmes (indépendamment de l'utilisateur ayant lancé le processus)
- Diffère d'autres systèmes MAC (par ex, SELinux) car les objets ne sont pas étiquetés
  - Fonctionne sur la base des chemins de fichiers

AppArmor

- Le confinement réalisé se base sur des profils chargés par le noyau, au démarrage ou pendant le fonctionnement
- 2 modes de chargement pour les profils
  - « enforcement » : applique réellement le confinement
  - « complain » : mode audit, remonte les blocages que le confinement pourrait engendrer
- les modes peuvent être mixés selon les profils
  - Souvent un profil cible un programme

AppArmor

- Profil : ensemble de règles de plusieurs types
  - Mode d'accès sur des fichiers
  - Utilisation de capacités
  - Accès au réseau
  - Montage de système de fichiers
  - Signaux
- Modèle en liste blanche : il faut autoriser explicitement les accès ou l'utilisation de capacité
  - « #include » permet d'inclure des règles pré-établies
- syntaxe détaillée : `man 5 apparmor.d`

AppArmor

- Exemple de profil (extrait de TCPDump) :

```
#include <tunables/global>
/usr/sbin/tcpdump {
  #include <abstractions/base>
  ...
  capability net_raw,
  ...
  network raw,
  ...
  # for -D
  capability sys_module,
  @{PROC}/bus/usb/ r,
  @{PROC}/bus/usb/** r,
  @{HOME}/** rw,
  ...
}
```

Programme ciblé

Capacités

Accès réseau

Mode d'accès aux fichiers (@{PROC} et @{HOME} sont des variables prédéfinies)

SECCOMP

- Objectif : rendre possible l'exécution de code « étranger » depuis un processus de confiance, sans compromettre la sécurité du système
  - Contrôle des appels systèmes autorisés (*secure computing mode*)
  - prctl(PR_SET_SECCOMP, <mode>)
    - Nouvel appel système seccomp() depuis noyau 3.7
- « Mode 1 » (SECCOMP_SET_MODE_STRICT)
  - depuis le noyau 2.6.12
  - définit un mode d'exécution restreint
    - 4 appels système : read(), write(), exit() et sigreturn()
    - Pas de retour arrière possible

SECCOMP

- « Mode 2 » (SECCOMP_SET_MODE_FILTER)
  - depuis le noyau 3.5
  - définition d'une liste d'appels système (et les arguments autorisés) décrite avec des filtres BPF (*Berkeley Packet Filter*) : seccomp-bpf
  - Filtre transmis aux processus et *threads* enfants
  - NO_NEW_PRIVS bloque la modification de la liste
- Vérifier l'utilisation pour un processus
  - grep Seccomp /proc/<pid>/status
    - 0 : Seccomp n'est pas activé
    - 1 : Seccomp est activé en mode strict
    - 2 : Seccomp-bpf est activé

PatchGuard et Code Integrity

- Windows Kernel Patch Protection (« Patch Guard »)
  - Protection contre certaines techniques de modification du noyau (code ou structures de données) utilisées par les *rootkits*
  - Surveille notamment la SSDT (*System Service Dispatch Table*), l'IDT, la GDT, certains pointeurs de fonctions du noyau
  - Autoprotection
    - Mise en œuvre des techniques anti-debug
    - Chiffrement de ses structures de données
  - Interrompt le fonctionnement du système (BSOD) en cas de détection d'un comportement malveillant

PatchGuard et Code Integrity

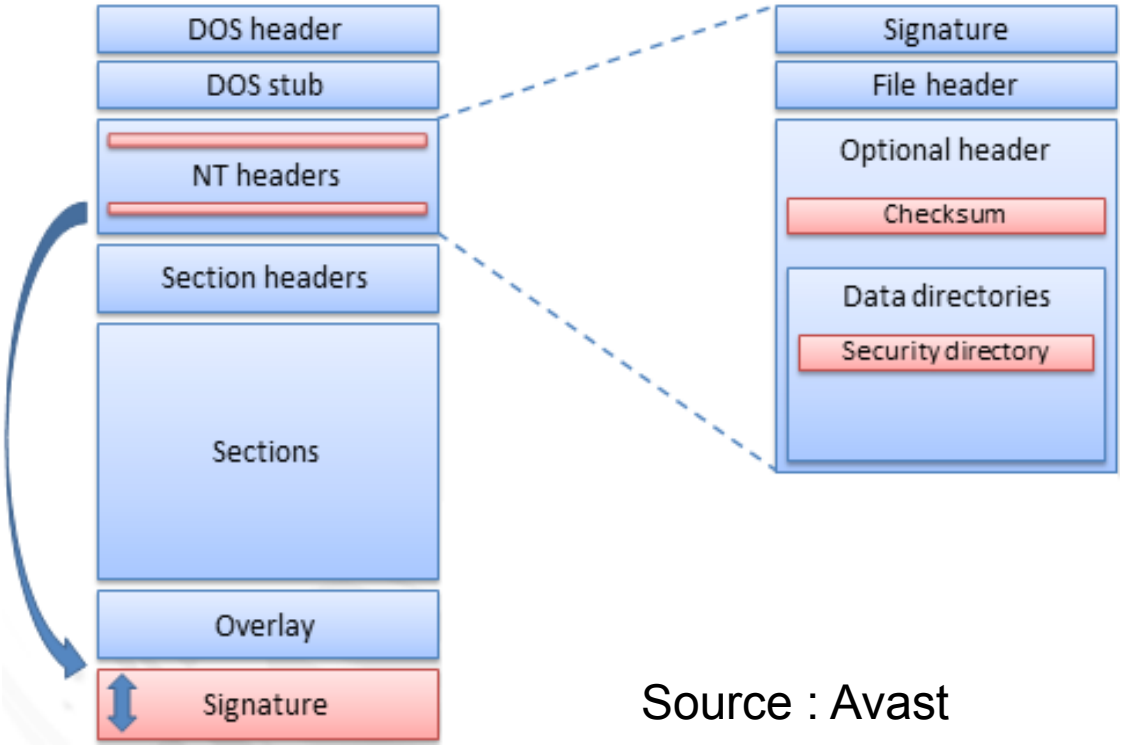
- KMCI : *Kernel-Mode Code Integrity*
  - Signature des pilotes
    - fichier exécutable de type binaire PE
    - extension .sys
  - le code d'un pilote s'exécute avec les mêmes privilèges que le reste du noyau
- Contrôle du chargement des pilotes
  - Sur les versions 64 bits, le pilote doit être signé :
    - Avant Windows 10 1607, par une autorité de certification reconnue par Microsoft
    - Depuis Windows 10 1607, en principe par Microsoft
      - dépend de la date de signature du pilote

PatchGuard et Code Integrity

- UMCI : *User-Mode Code Integrity*
  - Signature des programmes (.exe), des bibliothèques (.dll) et des ActiveX (.ocx)
    - Signature incluse au binaire ou utilisation de catalogue (collection signée de hash)
  - Par défaut, la signature de code n'est pas obligatoire
    - Activable (*) depuis Windows 10 via Device Guard/Windows Defender Application Control
  - Le contrôle de la signature se fait sur la base du magasin de certificat du système
    - Ajout d'autorité de certification possible (contrairement à KMCI)

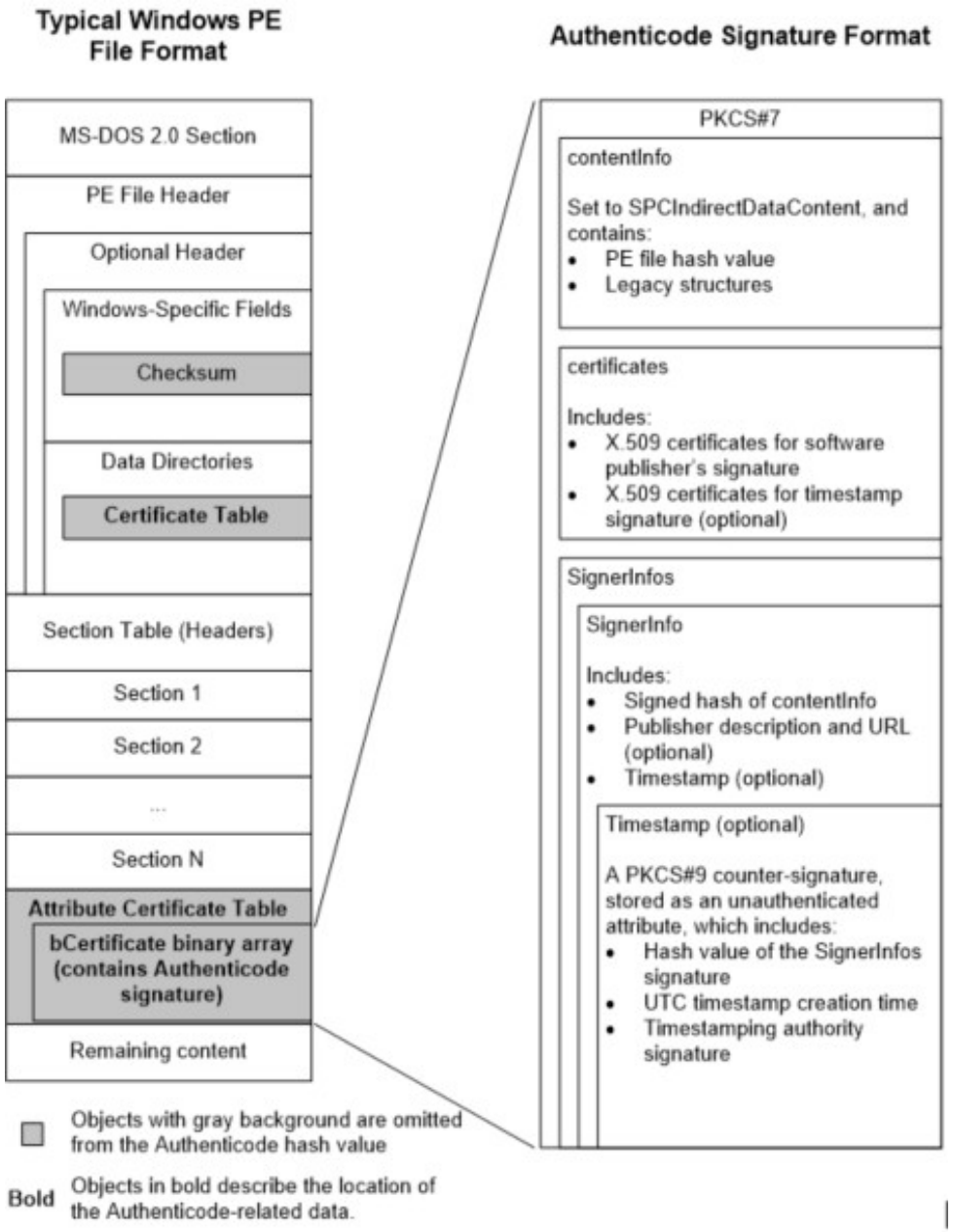
PatchGuard et Code Integrity

- Signature d'un binaire Windows (format PE)
  - Inclusion d'une signature dite « Authenticode » au format PKCS#7 stockée dans un certificat X509
  - Signature contient le haché des parties en bleu
  - Algorithme de hachage : MD5, SHA1, SHA256
  - Vérification de la signature implique de vérifier la chaîne de certification (i.e. remonter jusqu'à l'autorité de certification racine)



Source : Avast

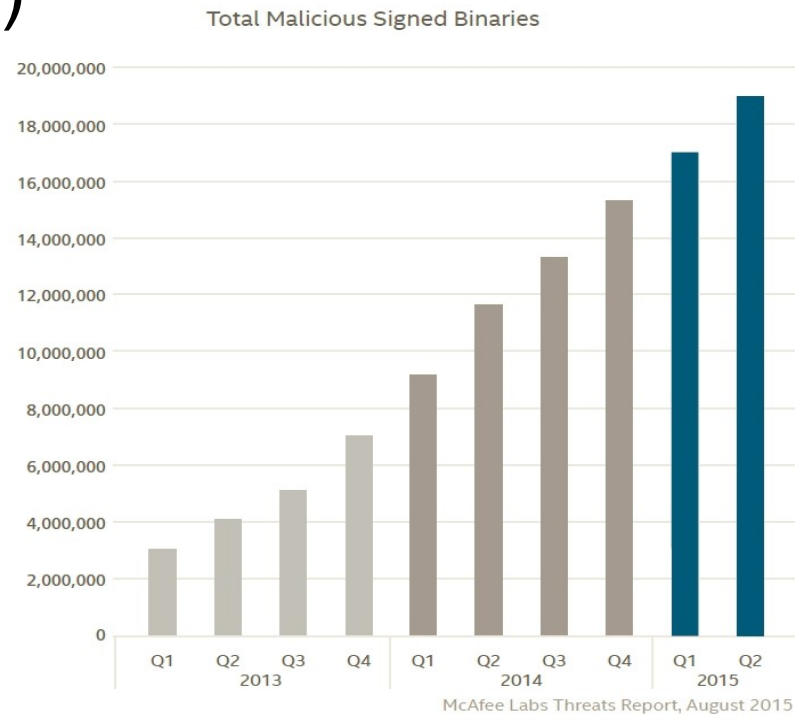
PatchGuard et Code Integrity



Source : Microsoft

PatchGuard et Code Integrity

- Limite de la signature
  - Un code signé peut être utilisé à des fins malveillantes (par ex, LoJax avec le pilote RwDrv)
  - Pas de garantie sur la sécurité du code
  - De plus en plus de *malware* sont signés (vol de certificats et de leur clé privée)
    - Identification du vol souvent tardive
    - Déclaration du vol pour sa révocation
    - Diffusion de la révocation des certificats délicate



Protection Windows

- Mécanismes de protection incorporés au système d'exploitation
  - Processus protégés
    - Mis en place depuis Windows Vista pour le DRM
      - Évolution dans Windows 8.1, notamment pour protéger LSASS
    - Impose l'utilisation de code signé (*UMCI*)
    - Processus non-accessibles avec un *debugger* conventionnel
  - Antivirus *Windows Defender* intégré nativement depuis Windows 8
    - Le moteur d'analyse peut fonctionner dans un bac à sable (nécessite Windows 10 1703 ou supérieur)

Protection Windows

- Mécanismes de protection incorporés au système d'exploitation (suite)
  - *Windows Defender Exploit Guard* depuis Windows 10 1709
    - Permet d'appliquer des techniques anti-exploitation (*mitigation*) aux processus
      - Bloquer le chargement de dll distantes, désactiver la création de processus enfant, désactiver certains appels systèmes, filtrer les adresses d'import ou d'export
    - Peut fonctionner en mode audit pour tester l'impact des mesures
    - Ajoute des capacités à Windows Defender AV
      - notamment pour limiter l'exploitation de vulnérabilités dans la suite Office, certaines exécutions de commandes à distance ou le vol d'éléments d'authentifications dans LSASS

AppLocker

- Mécanisme permettant de restreindre l'utilisation d'applications à une liste pré-définie (approche par liste blanche)
  - Permet de bloquer certaines applications malveillantes
  - Peut empêcher l'exécution d'application dans des versions obsolètes et/ou vulnérables
- Disponible depuis Windows 7 version Enterprise et depuis Windows Server 2008 R2
  - Supplée les politiques de restriction logicielle (SRP) apparues dans Windows XP (qui restent disponibles)
  - Différencie les utilisateurs (contrairement à SRP)

AppLocker

- Cible :
  - fichiers exécutables (com, exe) et bibliothèques (dll, ocx) ;
  - programmes d'installation (msi, msp, mst) ;
  - scripts (ps1, bat, cmd, vbs, js) ;
  - depuis Windows 8, les applications « universelles » (appx)
- Pour identifier un fichier, peut se baser :
  - sur le nom de l'éditeur (au niveau du certificat de signature) ;
  - Sur un chemin;
  - Sur un condensat cryptographique (hash)

Pour approfondir

- Intel® 64 and IA-32 Architectures Software Developer's Manual (Vol. 3A, ch. 3 à 6)
- AMD64 Architecture Programmer's Manual (Vol. 2, ch. « Segmented Virtual Memory » et « Page Translation and Protection »)
- Eric Lacombe, Thèse sur la sécurité des noyaux de systèmes d'exploitation, 2009
- Ken Johnson et Matt Miller, Exploit Mitigation in Windows 8, Black Hat USA 2012
- Andrea Allievi, Understanding and defeating Windows 8.1 Kernel Patch Protections, NoSuchCon 2014

Pour approfondir

- Matthew Jurczyk and Gynvael Coldwind, GDT and LDT in Windows kernel vulnerability exploitation, 2010
- Julien Tinnes and Chris Evans, Security In-Depth for Linux software, HITB Malaysia 2009
- Daniel Gruss *et al.*, KASLR is Dead: Long Live KASLR, 2017
- Igor Glücksmann, Injecting custom payload into signed Windows executables, REcon 2012
- Alex Ionescu, Unreal Mode : Breaking Protected Process, NoSuchCon 2014

Questions ?