Master 2 SeCReTS 2017-2018
Module Sécurité Système
Examen

# Sécurité des systèmes Unix

**Merci de bien lire les consignes :**
— une seule feuille A4 manuscrite autorisée ;
— aucune communication ;
— aucun accès à un ordinateur, une station de travail, un téléphone portable, une calculatrice, un PDA ou tout autre dispositif électronique, connectable ou non ;
— sujet à remettre en fin d'examen ;
— n'oubliez pas d'**indiquer nom et prénom sur la copie**.

*Un point est attribué au soin apporté à la rédaction des réponses.*

## Première partie

# Sécurisation intrinsèque du système (10 pts)

## Questions de cours (4pts)

1. *(0.5pt)* En quoi une négligence sur la sécurité physique d'un système peut réduire à néant tous les efforts faits sur la sécurité applicative de celui-ci ? Citez un exemple.
2. *(0.5pt)* Le terminal d'accès ultra-sécurisé d'une entreprise soucieuse de préserver ses brevets dans le domaine aéronaval demande à ses utilisateurs de s'authentifier grâce à un mot de passe de 16 caractères. Pour accéder au sous-réseau confidentiel, l'application demande en plus à l'utilisateur d'entrer un code PIN à 8 chiffres. S'agit-il d'une authentification dite « forte » ? Justifiez.
3. *(1.0pt)* Le mécanisme de mise en route du moteur d'une automobile futuriste est géré par un système embarqué basé sur une variante minimaliste d'Unix, qui implémente les PAM. Pour démarrer le véhicule, il est nécessaire que l'utilisateur soit sobre (vérification faite par `pam_ethylo.so`) et qu'il possède des points sur son permis de conduire (vérification faite par `pam_licensepoints.so`). Une fois ces vérifications faites l'utilisateur doit valider son empreinte digitale (vérification faite par `pam_digitalprint.so`) ou son empreinte rétinienne (vérification faite par `pam_retinalprint.so`). Après cette dernière validation le véhicule démarre. Rédigez la configuration du module.
4. *(0.5pt)* Voici un extrait de fichier `/etc/passwd` :

```
root:x:0:0:root:/root:/bin/bash
romain:x:1000:1000:romain:/home/romain:/bin/bash
sync:x:800:800::/tmp:/sbin/sync
william:x:1001:1001:william:/home/william:/bin/zsh
```

   Donnez le nom du compte captif, et expliquez brièvement à quoi il sert.

5. *(1.5pt)* Kevin souhaite mettre en place un *reverse-proxy* web sur son réseau personnel, et choisit d'installer un système dit « bastion ». Il opte pour la distribution Linux *Debian*. Il réalise un inventaire rapide des *packages* installés. Lesquels ne conviennent pas ? Pour chacun, expliquez pourquoi.

   — `acpid` : daemon de gestion des évènements de l'alimentation
   — `base-files` : divers fichiers de base du système Debian
   — `bash` : shell utilisateur

— `bluez` : drivers et outils pour la gestion du bluetooth
— `bsdgames` : jeux en mode console
— `firefox` : navigateur web
— `gdb` : GNU debugger
— `gcc` : compilateur C
— `initscripts` : scripts de lancement et d'arrêt du système
— `iptables` : outils d'administration pour le filtrage réseau
— `irssi` : client basé sur un terminal pour le protocole IRC
— `lvm2` : gestionnaire de volumes logiques Linux
— `mount` : outils pour monter les systèmes de fichiers
— `nmap` : simple scanner réseau
— `ntp` : daemon de synchronisation de l'horloge via le réseau
— `sox` : outils divers de manipulation et tranformation du son
— `ssh` : metapackage client et server pour le shell sécurisé
— `sudo` : outil de délégation de privilèges aux utilisateurs
— `tcpdump` : outil de surveillance et de capture réseau
— `udev` : daemon de management des périphériques
— `xorg` : serveur graphique sous linux

## *Don't mess with myman* (6pts)

L'objet de cet exercice porte sur l'exploitation d'un binaire nommé `myman`, un programme analogue à la commande `man` bien connue sur les systèmes Unix.

1. *(0.5pt)* Voici ce qu'on peut lire dans la console en essayant d'utiliser `myman` :

```
$ myman -h
usage: myman -h
       myman <fichier>
Affiche le contenu de /var/doc/<fichier> grâce à l'outil less.
$ myman ../../etc/debian_version
7.7
```

Expliquez ce résultat. À quel type de faille est visiblement vulnérable cet outil ?

2. *(0.5pt)* Quel mécanisme mettriez-vous en place pour y remédier ?
3. *(1.0pt)* L'outil de mise en page est `more`.

```
$ ldd $(which less)
  linux-vdso.so.1 =>  (0x00007fff7e5ff000)
  libtinfo.so.5 => /lib64/libtinfo.so.5 (0x00007f05dc597000)
  libc.so.6 => /lib/libc.so.6 (0x00007f77d4e82000)
  /lib64/ld-linux-x86-64.so.2 (0x00007f77d5447000)
```

Tracez l'arborescence du système de fichiers qu'il sera nécessaire de construire pour mettre en place le mécanisme évoqué à la question précédente. Expliquez la présence de chaque noeud de l'arbre que vous donnerez.

4. *(1.0pt)* Quelles limitations présente le mécanisme que vous avez mis en place ? Quelle solution voyez-vous pour vous prémunir au mieux de la vulnérabilité ?
5. *(0.5pt)* Voici ce qu'on peut lire après un nouvel essai :

```
$ myman attention_ce_paramètre_est_le_plus_long_paramètre_du_monde_AAAA
Segmentation Fault
$ dmesg | grep segfault
[1337.314159] myman[4242]: segfault at 41414141 sp ffffd3a4 error 14
```

Expliquez ce résultat. À quel type de faille est visiblement vulnérable cet outil ?

6. *(1.0pt)* Dessinez soigneusement la pile au moment du crash (sur un processeur x86) sur lequel vous ferez ressortir (en le faisant apparaitre en rouge par exemple) l'élément que l'on va chercher à maîtriser pour exploiter la vulnérabilité. Vous devez faire apparaitre les éléments suivants, au minimum :
   — une flèche montrant le sens de croissance des adresses ;
   — une flèche montrant le sens de croissance de la pile ;
   — les sauvegardes diverses (adresse de retour, base de pile) ;
   — les zones contenant des variables locales ;
   — le contenu des registres ESP et EBP.
7. *(1.0pt)* Quelles protections pourraient compliquer l'exploitation de cette faille ? Pour chacune de ces protections, voyez-vous un moyen de contournement ?

## Culture générale (1pt bonus)

Citez un évènement public en rapport avec la sécurité informatique paru ces derniers jours (conférence, publication, parution d'outil, d'exploit, de vulnérabilité, une attaque ciblée, etc.). Donnez quelques détails sur ce sujet en particulier. Si votre réponse est très satisfaisante, elle peut donner lieu à des points hors barème.

### Deuxième partie

# Prévention et confinement des attaques (10 pts)

1. *(1pt)* Sur la prévention des attaques :
   — Expliquez brièvement quel principe est utilisé pour protéger un programme contre les buffer *overflow* dans la pile au moment de la compilation.
2. *(2pts)* Un attaquant possède les informations suivantes sur le système qu'il cible : la pile des programmes est non-exécutable (NX), mais la bibliothèque C standard (`libc`) est toujours chargée à la même adresse.
   — Comment l'attaquant peut exploiter un *buffer overflow* dans la pile d'un programme vulnérable avec ces informations ?
   — Quelle fonctionnalité du noyau l'administrateur du système ciblé devrait activer pour éviter qu'une attaque réussisse ?
3. *(3.5pts)* Conteneurs et virtualisation
   — *(1pt)* Expliquez la différence fondamentale entre les mécanismes de virtualisation (comme VirtualBox) et les mécanismes de conteneurs (comme LXC ou Docker).
   — *(0.5pt)* Quels sont les noms des 2 mécanismes qui permettent de créer des conteneurs sous Linux ?
   — *(1pt)* Donnez 2 cas d'usage de la virtualisation sous Linux.
   — *(1pt)* Décrivez la démarche permettant de restreindre le nombre de CPU utilisables par un groupe de processus.
4. *(3.5pts)* Mandatory Access Control
   — *(1pt)* On s'intéresse à l'idée de coupler le mécanisme de `chroot` à un mécanisme de *Mandatory Access Control* (MAC) comme SELinux.
   — Expliquez en quoi SELinux est complémentaire du mécanisme `chroot`.
   — Dans ce contexte, décrivez brièvement un schéma d'attaque qui serait bloqué par SELinux.
   — Observez ce listing qui représente un profil AppArmor :

```
1  # Declare an apparmor variable to help with overrides
2  @{MOZ_LIBDIR}=/usr/lib/firefox
3
4  #include <tunables/global>
5
6  /usr/lib/firefox/firefox {
7    #include <abstractions/audio>
8    #include <abstractions/cups-client>
9    #include <abstractions/dbus-strict>
10   #include <abstractions/dbus-session-strict>
11   #include <abstractions/dconf>
12   #include <abstractions/gnome>
13   #include <abstractions/ibus>
14   #include <abstractions/nameservice>
15   #include <abstractions/openssl>
16   #include <abstractions/p11-kit>
17
18   # for networking
19   network inet stream,
20   network inet6 stream,
21
22   /etc/ r,
23   /etc/mime.types r,
24   /etc/mailcap r,
25   /etc/firefox*/ r,
```

```
  /etc/firefox*/** r,

  deny @{MOZ_LIBDIR}/** w,
  deny /usr/lib/firefox-addons/** w,

  owner @{HOME}/ r,
  owner @{HOME}/Public/ r,
  owner @{HOME}/Public/* r,
  owner @{HOME}/Downloads/ r,
  owner @{HOME}/Downloads/* rw,
  owner @{HOME}/.{firefox,mozilla}/ rw,
  owner @{HOME}/.{firefox,mozilla}/** rw,
  owner @{HOME}/.{firefox,mozilla}/**/*.{db,parentlock,sqlite}* k,
  owner @{HOME}/.cache/mozilla/{,firefox/} rw,
  owner @{HOME}/.cache/mozilla/firefox/** rw,
  owner @{HOME}/.cache/mozilla/firefox/**/*.sqlite k,

  # Addons
  #include <abstractions/ubuntu-browsers.d/firefox>

  # Site-specific additions and overrides. See local/README for details.
  #include <local/usr.bin.firefox>
}
```

— *(0.5pt)* Quelle est l'application concernée par ce profil AppArmor ?
— *(2pts)* Pour chaque bloc de règles en gras, décrivez la fonctionnalité concernée de l'application (on ne demande pas de détailler les règles une par une).

---

**Troisième partie**

# Test et fuzzing (5pts)

## Test (3pts)

Soit la pseudo fonction suivante :

```
f(arg) {
  if (arg > 10)
      print "Sup"

  if (arg % 2 == 0)
      print "pair"
}
```

1. *(1.5pts)* Écrire les jeux de tests pour chacun des critères suivants, en minimisant le nombre de test :
   (a) *code coverage*
   (b) *branch coverage*
   (c) *path coverage*
2. *(1.5pts)* Donnez 2 exemples de mutants :
   (a) un tué par les tests de *code coverage*
   (b) un tué par les tests de *branch coverage* mais pas par ceux de *code coverage*

   Les mutants sont considérés comme tués s'ils ne donnent pas la même sortie que le programme original.

## Fuzzing (2pts)

Soit un programme que l'on sait accepter pour entrée "echo123".

3. *(2pts)* Proposez 2 mutations qui pourraient potentiellement lever des bugs (et justifiez-les brièvement).

---

**Fin de l'examen.**

---