Introduction à la sécurité informatique

Table des matières

1 Notions de base de la sécurité

1.1 Confidentialité, Intégrité et Disponibilité

- Confidentialité : Une ressource ne peut pas être accédée par des tiers non autorisés
- Intégrité : Une ressource ne peut pas être modifiée ou supprimée par des tiers non autorisés
- Disponibilité : Une ressource (ou un service) peut être accédée dans un temps raisonnable quand l'utilisateur en a besoin

1.2 Moindres privilèges et Séparation des privilèges

- Moindres privilèges : Un processus s'exécute avec le minimum de privilèges dont il a besoin pour mener à bien sa tâche
- Séparation des privilèges : Plusieurs processus, avec des privilèges différents, coopèrent pour mener à bien une tâche. Un processus de moindre privilège peut demander à un processus plus privilégié d'effectuer des tâches pour lui

2 Failles classiques

Les principales failles sur des programmes sont de type détournement d'exécution vers du code choisi par l'attaquant :

- Débordements de tampons
- Chaînes de format

2.1 Débordements de tampons

Lors de l'appel d'une fonction par une autre, dans la pile d'exécution sont empilés :

- Les paramètres de la fonction appelée
- L'adresse de retour dans la fonction appelante
- Les variables locales de la fonction appelée

Dans la fonction appelée, quand une variable locale « déborde », par exemple recopie (avec **strcpy()**…) d'une longue chaîne dans un tampon trop petit, l'adresse de retour est modifiée.
Habituellement quand la fonction appelée s'achève, une erreur de type "**Segmentation fault - core dump**" met fin au programme.
En choisissant les données envoyées, l'attaquant modifie l'adresse de retour pour pointer vers son propre code qui est alors exécuté par le programme. Ce code choisi est appelé **shell-code** car il correspond habituellement à **exec("/bin/sh")** et l'attaquant se retrouve alors dans un shell.

2.2 Chaînes de format

Les fonctions de mise en forme de texte nécessitent une chaîne de format, pour spécifier le type et le format d'affichage de chaque donnée.
Si le programmeur ne spécifie pas cette chaîne alors le premier paramètre est utilisé comme chaîne de format.
Quand il s'agit d'une donnée externe alors :

- La pile d'exécution peut être scannée **(%<val>$x)**
- Et peut être modifiée **(%n)** dont l'adresse de retour

2.3 Failles des applications WEB

- Injections (SQL, de commande…) : changement de sémantique d'une requête SQL, d'une commande système…
- Authentification et gestion des sessions défaillantes
- XSS (Cross Site Scripting) : exécution de code sur le navigateur de la victime via un serveur de confiance
- XSRF : un attaquant fait en sorte que le navigateur de la victime envoie des requêtes au serveur
- Vol de session / d'identité : usurpation de l'identifiant d'une session / d'un utilisateur
- Directory traversal : accès hors de l'arborescence d'origine

2.4 Autres types de failles

Enfin de nombreuses autres failles peuvent avoir lieu quelque soit le type de programme :

- Épuisement de ressources (mémoire, disque, CPU…)
- Débordement d'entier
- Séquences non atomiques (TOCTOU)
- Dénis de service (DoS) : aux niveaux réseau et applicatif..

3 Inventaire et publication de failles

3.1 Publication de failles

Quand un éditeur corrige une faille de sécurité, il va publier un avis de sécurité, généralement dans une liste de diffusion dédiée et un site web dédié.
Grâce à cet avis de sécurité, les utilisateurs vont pouvoir prendre connaissance des informations suivantes :

- versions de logiciels vulnérables et version non vulnérables
- type de faille
- conséquence de l'exploitation de la faille

Certaines failles de sécurité peuvent être diffusées par plusieurs éditeurs. Notamment quand un logiciel standard (apache, postfix…) est vulnérable, à la fois son éditeur, mais également les éditeurs des systèmes d'exploitation (debian, RedHat, FreeBSD…) qui utilisent ce logiciels , vont diffuser leurs propres avis de sécurité.

3.2 Inventaire de failles

Des services, gratuits ou payants, permettent d'inventorier les failles au fur et à mesure de leurs publications. L'un des services les plus connus est le **CVE** (Common Vulnerability Exposure) qui permet également d'attribuer une référence unique à différents avis de sécurité traitant de la même faille de sécurité.
D'autres services sont les **CERT**. Un de leurs services est de redistribuer des avis de sécurité, mais restreints à ceux qui concernent chaque client, tout en les personnalisant notamment au niveau de la gravité de leurs conséquences en cas d'exploitation.
Références :

- https://cve.mitre.org/

- https://www.us-cert.gov/

4 Notions de sécurité sous Windows

4.1 Processus et objets

Chaque processus est composé d'au moins un thread, de type utilisateur, service, système ou sous-système (posix). Les processus sont visibles avec Process Explorer de la suite Sysinternals (Microsoft).
Chaque objet a un nom dans une arborescence, par exemple les fichiers dans l'arborescence des répertoires.
Un processus accède à un objet via un descripteur. Les objets sont visibles avec WinObj de la suite Sysinternals (Microsoft).

4.2 Base de registres

La base de registres contient les configurations statiques et dynamiques du système et des programmes sous la forme d'une arborescence de clés et de valeurs.
La base de registres est modifiables par regedit (regedt32), mais aussi par de plus en plus d'interfaces disponibles comme mmc.
Les registres sont visibles avec Registry Monitor de la suite Sysinternals (Microsoft)

4.3 Comptes et groupes

Les comptes font partie de 2 catégories : utilisateurs et machines (finis par '$'). Ils sont identifiés par un nom et un **SID**. Ils sont définis dans la **SAM** (local) ou l'**AD** (Active Directory)

- **net user « Administrateur »**

Les groupes sont un regroupement de comptes et de groupes. Ils permettent d'accorder des droits sur des ressources d'après une utilisation identique de ces ressources.

4.4 Domaine et contrôleur de domaine

Un domaine est un espace de confiance. Le contrôleur de domaine (DC) gère la base des comptes (utilisateurs et machines) d'un domaine. Un domaine peut avoir plusieurs DC synchronisés.
Le **SID** d'un compte est toujours de la forme « **S-1-5-21-XXX-XXX-XXX-N** » où « **XXX-XXX-XXX** » est l'identifiant du domaine et **N** est l'identifiant du compte dans le domaine.

4.5 Authentification et autorisation

Plusieurs mécanismes d'authentification sont disponibles : **LanMan**, **NTLM**, **Kerberos**…
Lors de l'ouverture d'une session est créé un **contexte de sécurité** (**jeton**) qui est complété par la **LSA locale** et la **LSA de domaine**, avec :

- l'identité de l'utilisateur,
- les identités des groupes dont fait partie directement ou indirectement l'utilisateur et
- les privilèges accordés à l'utilisateur et à ces groupes.

Les privilèges sont les droits d'accès à certaines fonctions privilégiées (par ex : backup, restore…)
L'autorisation d'accès d'un processus à un objet se fait suivant le jeton du processus et les permissions de l'objet.

4.6 Permissions et audit

Les permissions d'un objet contiennent :

- Le **SID** du propriétaire

- Les **DACL** (Discretionary Access Control List), sont les permissions d'accès de type **ACL** (liste de : « compte/groupe + type d'accès + OK/KO ») définies par le propriétaire de l'objet

- Les **SACL** (System Access Control List), sont les audits d'accès (enregistrement des accès) de type **ACL** (liste de : « type d'accès + OK/KO + audit/pas audit ») définis par l'administrateur

4.7 Héritage

Un processus/thread hérite du jeton de son père.
Personnification : un processus/thread peut avoir un second jeton, par exemple dans un processus serveur, pour réaliser des tâches en tant que l'utilisateur, puis retourne à ses privilèges initiaux par destruction du second jeton.
Un objet hérite des permissions de son container, par ex un fichier de son répertoire. Il y a possibilité de bloquer l'héritage, ie l'objet part d'une liste vide de permissions.

5 Notions de sécurité sous Unix

5.1 Historique d'Unix

Unix est un **système ouvert** largement documenté (pages man).
Il a été standardisé par de multiples standards :

- 1988 : IEEE POSIX (POSIX.1)
- 1990 : X/Open (XPG3) et Open Software Fondation
- 1994 : Single Unix Specification (SUS)
- 1997 : Open Group de la fusion de X/Open et OSF
- 1999 : Collaboration de POSIX et SUS
- 2001 : Posix 2001 / SUSv3 unifie POSIX et Open Group

Le dernier standard est celui de 2008 : **Posix 2008** / **SUSv4**
Il est à noter que des systèmes non Unix, comme Windows, sont en partie compatibles avec Posix.
Les différents constructeurs effectuent systématiquement des tests d'inter-opérabilité avec les logiciels tiers.

5.2 Utilisateurs

Un utilisateur n'est qu'un numéro, nommé **UID**. Deux utilisateurs ne doivent pas avoir le même.
L'**administrateur** est le compte qui possède l'**UID zéro**, généralement appelé **root**. Aucun contrôle d'accès lui est appliqué.
Les utilisateurs peuvent être classés dans différentes catégories.

- Les utilisateurs standards ont un UID non nul.
- Les utilisateurs privilégiés ont la possibilité d'obtenir l'UID nul via un programme comme su (mot de passe partagé), sudo ou calife.
- Les utilisateurs dédiés ne font que posséder des fichiers et/ou exécuter un service, et ne possèdent pas d'accès interactif.

5.3 Groupes : principal et secondaires

Chaque utilisateur possède :

- un groupe principal qui est son groupe par défaut, par exemple lors de la création de fichiers…
- des groupes secondaires, ils permettent généralement d'autoriser ou d'interdire l'accès à des ressources.

5.4 Fichiers

5.4.1 Systèmes de fichiers

Le système de fichiers (FS) permet d'organiser les fichiers dans une arborescence hiérarchique. Un répertoire est un fichier listant des fichiers. Le FS associe les fichiers et leurs droits d'accès. Les partitions sont montées sur des points de montage : un répertoire d'une partition déjà montée. La racine de la première partition montée est la racine du système de fichiers.
Différents types de FS existent, comme :

- les FS locaux (ext2/3/4, reiserfs, jfs, xfs, sysv, ufs… sous Unix ; iso9660 (RR/joliet), udf, (v)fat, ntfs… par ailleurs),
- les FS virtuels (procfs, sysfs, tmpfs (RAM), fuse (UserLand)…) et
- les FS réseaux (nfs, coda, afs; non-Unix : smb/cifs, ncp…).

5.4.2 Fichiers

Un fichier est un **inode** (numéro) dans une partition. À l'inode sont associés les caractéristiques du fichier :

- Utilisateur et groupe propriétaires
- Droits d'accès
- Heures
  - De Modification du contenu
  - D'Accès au contenu
  - De Changement des métadata
- Nombre de liens

Deux principaux types de fichiers existent :

- Fichier ordinaire (-)
- Répertoire (d) : fichier dont le contenu est une liste de liens entre un nom de fichier et :
  - un inode : lien hard. Comptabilisé.
  - un autre nom de fichier : lien soft (l)

Pour effacer un fichier :

- si c'est un lien hard alors le nombre de liens est décrémenté, et s'il est nul alors l'inode est effacé
- l'entrée dans le répertoire est effacée.

5.4.3 Fichiers spéciaux

Différents types de fichiers spéciaux existent :
pipe et socket : espace mémoire de taille fixe de type FIFO. Ils sont utilisés dans le cadre de la communication inter-processus
caractère et bloc : associés à des pilotes de périphérique via un **couple majeur/mineur**, par exemples l'accès à une partition d'un disque dur, d'un clavier, d'un écran, de la mémoire, d'un générateur de nombres aléatoires… Si le nom d'un tel fichier n'a aucune importance, les droits d'accès à ces fichiers sont très importants, par exemple si une partition est en lecture pour tout le monde alors tous les fichiers qui y sont stockés seront accessibles à tous les utilisateurs.

5.5 Droits d'accès

Les droits d'accès de tout fichier est constitué de 3 groupes (**User**, **Group**, **Other**) de 3 droits (**Read**, **Write**, **eXecute**) :
Les programmes peuvent avoir un droit spécifique : **suid/sgid** (**s**). Un programme avec un de ces droits s'exécute en tant que l'utilisateur (**suid**)/le groupe (**sgid**) propriétaires au lieu de s'exécuter en tant que l'utilisateur/le groupe qui lance le programme.
Par exemple pour qu'un utilisateur puisse changer son mot de passe il faut passer par un programme suid root qui pourra alors modifier un fichier qui n'est accessible qu'à l'utilisateur root.
Note : ce droit se note « **s** » si le droit « **x** » est aussi présent, « **S** » s'il est absent.
Sous Unix pour accéder à un fichier, il faut aussi pouvoir accéder à tous les répertoires au dessus.
Les droits d'accès sur les répertoires sont interprétés légèrement différemment :

- r permet de lister les fichiers
- w permet de créer, renommer et supprimer un nom de fichier
- x permet d'accéder dans le répertoire
- sgid : les fichiers créés héritent du groupe du répertoire, et non pas du groupe principal de l'utilisateur.
- sticky bit (**t**) : l'utilisateur ne peut supprimer ou renommer que ses propres fichiers. Utilisé sur les répertoire temporaires tel que **/tmp**

Attention, les droits de renommage et de suppression d'un fichier ne dépendent pas des droits d'accès du fichier ; seuls les droits d'accès au répertoire contenant le lien est pris en compte.
umask permet de définir les droits non donnés par défaut lors de la création d'un fichier, souvent :

- 0022 (contrôle total pour l'utilisateur propriétaire, lecture et exécution pour le groupe et les autres),
- 0077 en mode « paranoïde » (contrôle total pour l'utilisateur propriétaire, aucun accès pour le groupe et les autres)

chmod permet de changer les droits d'accès

5.6 Processus

Un processus est une instance en mémoire d'une exécution d'un programme
Chaque processus a son propre numéro unique : **PID**. Le processus avec le **PID == 1** est le premier lancé par le noyau au lancement du système, nommé **init**, et doit gérer le mode d'exécution du système.
Chaque processus a un processus père : **PPID**
Si le processus père meurt ou le fils se détache alors **PPID == 1**, ie il est hérité par le processus init. Par exemple quand un service est lancé il se détache du shell qui l'a lancé.
Un processus possède plusieurs identités :

- uid/gid réel : l'identité de l'utilisateur qui a lancé le programme
- uid/gid effectif : si le programme est privilégié (**suid**/**sgid**) alors l'identité de l'utilisateur/le groupe propriétaire du programme.
- fsuid/fsgid : utilisé seulement pour accéder au FS (spécifique à Linux)

Voir « Setuid Demystified » de David Wagner (Usenix 02).

5.7 Authentification : mots de passe

L'authentification basique est réalisée via des mots de passe. Ces mots de passe sont hachés avec une fonction à sens unique, ce qui donne une empreinte. Il n'existe pas de fonction permettent de retrouver le mot de passe en clair à partir de l'empreinte.
Les fonctions de hachage utilisées peuvent être dérivée de DES, MD5, blowfish, SHA2… Quand une fonction de chiffrement est utilisée alors le mot de passe est utilisé comme clé de chiffrement pour chiffrer une

donnée connue.
Les empreintes sont enregistrées dans un fichier qui n'est accessible qu'à root.
Dans le cas d'une authentification réseau centralisée, un serveur **LDAP** est utilisé.

5.8 The Unix way of life : fonctions propres, fonctions particulières et autres fonctions

La version courte est :
« Un programme : une fonction »
La version longue est :
« Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface. »
Citation trouvée dans le chapitre « Basics of the Unix Philosophy » du livre « The Art of Unix Programming » par Eric Steven Raymond :
"the first fact to face is that unix was not developed with security, in any realistic sense, in mind..."
– Dennis M. Ritchie

5.8.1 Fonctions propres

La fonction **chroot()** permet de créer une cage en restreignant l'accès d'un processus à une sous-arborescence du système de fichiers. Attention : aucune autre fonction est restreinte. Le processus doit tourner **sous une identité dédiée non privilégiée**.
La fonction **jail()**, uniquement sous **FreeBSD**, permet de créer une cage qui restreint en plus les accès aux processus en dehors de la cage, même pour les processus privilégiés. La cage peut également avoir une adresse IP et un hostname dédiés.
Le système **docker**, uniquement sous Linux, permet de créer une cage avec une plus grande quantité de restrictions : RAM, CPU, FS, privilèges… Il est nécessaire de mettre les fichiers dans un container et par défaut seul le contenu du container est accessible. Il est également possible de personnaliser un des nombreux containers pré-définis : ubuntu, debian, centOS, fedora, httpd, nginx, mysql, postgresql, php, tomcat, openjdk, java, wordpress, drupal, joomla…

5.8.2 Mise en oeuvre de la séparation des privilèges

La séparation des privilèges est mise en place via deux processus :

- l'un avec des privilèges,
- l'autre sans privilège et restreint dans une cage.

- Le processus non privilégié est le seul à traiter des données externes et le seul à exécuter des fonctions complexes : cryptographie, compression…
- Le processus privilégié ne fait pas confiance au processus non privilégié. Dès qu'il reçoit une donnée mal-formatée alors il tue le processus non privilégié.

Les deux processus communiquent via des fichiers temporaires, des tubes, des segments de mémoire partagés…
Exemple d'implémentation dans OpenSSH, voir le lien suivant pour plus de détails :

- http://www.citi.umich.edu/u/provos/ssh/privsep.html

Le processus fils non privilégié, tourne sous l'identité **sshd** et est restreint à **/var/empty** . Il traite les données provenant de l'extérieur. Il effectue les fonctions complexes (cryptographie et compression).
Si une faille est exploitable, les privilèges obtenus sont restreints. Le processus père privilégié, fournit des infos via une socket (**fd**) et en récupère via un tube (**crypto**…) et un segment de mémoire partagée. Il vérifie l'authentification et lance un autre fils sous l'identité de l'utilisateur qui s'est correctement authentifié.

5.8.3 Fonctions particulières

Les **privilèges POSIX** (**1003.1e**) définissent un ensemble de privilèges unitaires présents sur un système. Par défaut un processus privilégié possède tous les privilèges unitaires alors qu'un processus utilisateur n'en possède aucun. Un processus fils hérite des privilèges unitaires de son père. Afin d'améliorer la sécurité, un processus privilégié peut ne pas avoir certains privilèges unitaires et un processus utilisateur peut avoir un ou plusieurs privilèges unitaires. Également, un processus père peut ne donner qu'un sous-ensemble de ses privilèges unitaires à ses fils.
Ces privilèges sont implémentés sous de nombreux systèmes **Unix** (**Solaris**, **Irix**, **Linux**…) mais ne sont que très peu utilisés, sauf dans un système de partitionnement tel que **docker**. Références :

- http://wt.tuxomania.net/publications/posix.1e/
- https://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.2/capfaq-0.2.txt

Les **secureLevels BSD** définissent un ensemble de niveaux, chacun désactivant un ensemble de privilèges unitaires du système. Il n'est pas possible de baisser de niveau et donc de réactiver un privilège unitaire. Il n'est pas possible de choisir quels privilèges unitaires désactiver ou ne pas désactiver.

5.8.4 Autres fonctions

Par défaut, les contrôles d'accès sont de type **DAC** (Discretionary Access Control) qui ne se base que sur les droits d'accès positionnés par l'utilisateur propriétaire de l'objet.
Certains systèmes peuvent avoir des contrôles d'accès de type **MAC** (Mandatory Access Control). Il sont positionnés par l'administrateur afin d'appliquer une politique de sécurité. Ces droits sont prioritaires sur les droits donnés par l'utilisateur.
Certains systèmes peuvent avoir des contrôles d'accès de type **ACL** (Access Control List). Il s'agit de d'une liste de droits s'appliquant à des utilisateurs et groupes pour accéder à un objet spécifique. Basé sur **POSIX 1003.1e** sur tous les **Unix** sauf **HP-UX** qui est compatible **Netware** et **Windows**.