Introduction à la sécurité – Cours 2

Catalin Dima

Attaques locales, et défenses

- Peuvent impliquer la prise de contrôle, même partielle, de la machine cible, ou un utilisateur légitime mais mal intentionné.
- Attaques avec des privilèges d'administateur : stopper le gestionnaire de services réseaux (`inetd`). tuer les processus, reconfiguration de système.
- Attaques sans ces privilèges : forker à l'infini, exploiter des failles de type buffer overflow.
- *Bombes logiques*
- Défenses contre les attaques locales :
  - Défenses contre les buffer overflow.
  - Distribution judicieuse des droits d'accès et des privilèges.
  - Faire tourner des testeurs d'integrite (*Tripwire*) – testeurs si certains fichiers critiques de configuration du système n'ont pas été altérés.

Attaques à distance

- Très populaires, car ne nécessitant la prise de contrôle à distance de la machine cible.
- Attaques par arrêt de service : basés sur des erreurs d'implémentation de TCP/IP :
  - *Land, Ping of death, Jolt2, Teardrop, Newtear, Bonk, Syndrop, Winnuke.*
  - *IP/DNS Spoofing, Smurf*, DoS distribué.
- Défenses : être à jour avec les patches correctifs, mais aussi se défendre contre les déguisements d'adresse.

Attaques et vulnérabilités

- Buffer overflow – problème du contrôle d'accès :
  - Mieux définir et gérer les droits d'accès.
  - Confinement des applications dans des *domaines de sécurité*.
- Spoofing (IP, DNS, autres) et man in the middle – problème de l'authenticité.
- Création de canaux cachés – problème de la fuite d'information.
  - Controler/spécifier les flux d'information.
  - Qui peut lire quoi et qui peut écrire où ?
- Déni de service par saturation de ressources – problème de largeur de bande.
  - Bien identifier les possibles goulots d'étranglement dans l'architecture des applications.

Structure de la mémoire d'un programme

- *Segment de code du programme.*
- *Données initialisées.*
- *Tas* – zone d'allocations de mémoire pour les besoins du programme.
  - Chaque allocation de mémoire récupère une nouvelle zone de mémoire libre dans cette zone.
- *Pile d'appels* : adresses de retour de fonctions, variables locales, arguments de fonctions...
  - Chaque appel à une fonction crée une série de nouvelles entrées dans cette pile.
  - Adresse de *retour* : adresse de l'instruction que le programme devrait exécuter après avoir terminé la fonction courante.
- Zone de *communication avec le noyau.*

La pile d'appels et le tas croissent en sense inverse l'une à l'autre.

Structure d'un appel de fonction

- Tout programme pourrait contenir des tâches identiques qui s'exécuteraient plusieurs fois au cours de la vie du programme.
- C'est le but des fonctions : morceaux de code qui peut être appelé dans plusieurs endroits dans un programme.
- Un processeur identifie l'instruction à exécuter dans son IP – pointeur d'instruction.
- Un appel de fonction se ferait alors par placement de son adresse dans l'IP.
- Problème quand la fonction se termine : comment faire connaître au processeur où doit-il revenir dans le programme principal ? !
- Solution : lors de l'appel, on pousse (`push`) aussi l'adresse de l'instruction suivante sur la pile !
- Alors, à la fin de la fonction, on récupère (`pop`) cette adresse de la pile, on la remet dans l'IP et le processeur pourra reprendre l'exécution de l'endroit d'ou la fonction a été appelée !

Structure d'un appel de fonction

- La pile ne sert pas qu'à la transmission des adresses de retour !

```
void exemple(char *chaine){
      char tampon[16];
      ..... // instructions ...
      ......
  }
```

- Tous les compilateurs modernes utilisent le passage de paramètres par la pile !
- Donc tout appel d'`exemple` entraînera aussi un push de 4 octets (taille d'un pointeur qq !) sur la pile.
  - En général, l'ordre d'empilage est d'abord les paramètres et ensuite l'adresse de retour.
- Non pas seulement les paramètres, mais aussi les variables locales se voient réserver de la mémoire sur la pile !
- Donc cet appel va réserver ... octets sur la pile.
  - N'oubliez pas la taille de l'adresse de retour !

Buffer overflow

Ou débordement de mémoire tampon

```
void exemple(char *chaine){
      char tampon[16];
      strcpy(tampon, chaine);
      return;
  }
void main(){
      char grd_tampon[256];
      int i;
      for(i=0; i<256; i++)
         grd_tampon[i]='a';
      exemple(grd_tampon);
  }
```

- La copie de la chaîne `grd_tampon` dans la zone pointée par la variable `tampon` débordera.
- L'adresse destination est l'adresse d'un *paramètre de fonction.*
- Le débordement écrasera certaines des valeurs sauvegardées sur la pile
- En particulier, l'**adresse de retour**.
- Donc, lorsque le processeur va dépiler les 4 octets qui lui indiquent où sauter, il va déconner...

Attaque *buffer overflow*

- Supposons que le code de la fonction est maintenant

```
void exemple(){
        char tampon[16];
        gets(tampon);
        return;
  }
```

- Et que celui-ci fait partie d'une commande du système d'exploitation !
- Alors un môsieu malveillant pourrait "saisir", à la place d'une chaîne de caractères inoffensive, du code machine malveillant (caractère par caractère !)
  - Une démande (`exec`) d'exécution d'un shell (UNIX).
  - Lancement d'une DLL (Windows).
  - Pas besoin de vraiment saisir – utiliser la redirection de l'entrée std !
- Code malveillant dépendant de la machine cible de l'attaque :
  - Le code machine n'est pas portable !
  - Le code malveillant peut lui-même être écrit et compilé pour un certain système d'exploitation.

Anatomie d'une attaque *buffer overflow*

- L'endroit où la pile d'appels se trouve est géré de manière dynamique.
- La dimension exacte de l'espace mémoire entre la zone de début du buffer overflow et la zone où on peut mettre du code qui ne provoquera pas d'erreur d'exécution n'est pas prédéfinie.
- L'espace disponible (pour placer le code malveillant) pourrait ne pas être suffisant.
- L'attaquant doit déviner ces informations, ou...
- ... peut utiliser du code machine avec plein de `NOP` pour améliorer les chances que l'adresse de retour tombe sur son propre code malveillant.
- ... ou bien peut utiliser des fonctions de bibliothèque qui balancent des programmes lui permettant de prendre le contrôle de la machine.

Attaque et défense

Attaques :

- Création de trappes
  - Nécessite de trouver une faille de type buffer overflow dans un programme qui tourne avec les privilèges (SUID) `root` ou administrateur Windows.
- Passer à la phase d'attaque de l'intérieur du système :
  - Attaques contre les mots de passe.

Défense :

- Espace de données non-exécutables.
- Detection de séquences trop longues de `NOP`s.
- Identification de signatures (angl. *signature matching*) :
  - "Sniffer" le traffic sur le réseau
  - Chercher et identifier certains modèles de paquets
  - Pouvant provenir d'une attaque buffer overflow.

Prévention des attaques buffer overflow

- D.p.d.v. des administrateurs de système :
  - se tenir au courant avec les derniers "patchs" correctifs de chez Windows, Red Hat, ...
  - Controler de manière le plus étroitement possible, le traffic réseau.
  - Configurer le système de sorte que la pile soit *non-exécutable*.
  - Système de contrôle d'accès – minimiser l'étendue de la pénétration.
- D.p.d.v. des programmeurs :
  - Éviter les fonctions C permettant l'attaque *buffer overflow* : `gets`, `fgets`, `memcpy`, `scanf`, `strcpy`...
  - Vérificateurs automatiques de code recherchant des sources possibles de code vulnerable aux buffer overflows : *ITS4*, *SLINT*...
  - Rendre plus difficile la modification de la pile par des attaques : *Stack Guard*, *Stack Shield*.

Spoofing (Angl. parodier)

- Imposture réussie d'un attaquant qui fait croire à un agent légitime que des données qu'il (l'imposteur) émet ou falsifie sont de confiance.
  - Peut avoir comme conséquence la prise de contrôle d'une connexion réseau, voire même d'une machine.

  Types de spoofing :
  - ARP Spoofing : faire croire qu'une adresse IP est associée à une autre adresse MAC que celle qui normalement lui est attribuée dans le réseau.
  - IP Spoofing : faire croire à un agent qu'il communique avec la machine ayant une certaine adresse IP, alors qu'il communique avec une machine malveillante qui forge les paquets TCP/IP.
  - DNS Spoofing : faire croire à un agent qu'un nom de domaine correspond à une adresse IP, alors que c'est faux.
- Peut permettre
  - Des détournements de sessions de communication, la faisant transiter par la machine de l'attaquant pour *sniffer*.
  - Des attaques de type *man in the middle*, ou des messages forgés par l'attaquant pourraient être injectés dans la communication.
  - Des impostures, pour se faire passer e.g. pour des serveurs et récupérer des informations sensibles.

Détails de spoofing

- ARP spoofing :
  - Empoisonnement des caches ARP dans un réseau switché Ethernet.
  - Envoi de paquets ARP forgés à la victime
  - ... en même temps que la mise hors ligne du posseseur légitime de l'adresse IP,
  - ... ou l'empoisonnement du cache ARP du réseau.
- IP spoofing :
  - Forger des paquets IP ayant dans le champs "source" une adresse IP qui n'est pas l'adresse de la machine source.
  - La communication duplex peut se faire en trafiquant aussi les "routes".
  - Peut servir à masquer des commandes malveillantes destinées à la gestion à distance d'une machine compromise.
- DNS spoofing :
  - Remplacement frauduleux d'une association nom-adresse IP.
  - Peut se faire par l'empoisonnement du cache DNS d'un DNS
  - ... ou en envoyant une réponse DNS forgée lors d'une demande légitime de la victime.
  - ... ce qui pourrait aussi impliquer des attaques de type ARP spoofing contre la victime ou DoS contre le DNS.

Contrer le spoofing & co.

- Filtrer aux bords du réseau les adresses IP privées sur les connexions entrantes.
  - Se fier uniquement au NAT !
  - Faire de même pour les paquets sortants !
- Ne pas utiliser le faible niveau d'authentification fourni par TCP/IP.
  - Faire attention au HTTP, qui repose lourdement sur l'authentification par IP !
- Sécuriser le DNS (mais gare aux attaques DoS).
- Protocôles d'authentification – Kerberos, etc.

Deni de service (DoS)

Prévenir les utilisateurs légitimes de leur accès aux ressources du système cible.

– Potentiel important de destruction.

– Très souvent menées sans grande difficulté technique.

Catégories d'attaque :

- Attaque local :
  - Par arrêt de services : tuer des processus ou les faire "crasher", reconfigurer le système,...
  - Par épuisement de ressources : forker à l'infini, remplir le système de fichier.
- Attaque à distance :
  - Par arrêt de service : envoyer des paquets mal formés.
  - Par épuisement de services : attaque de type *SYN flood*,...

Attaques à distance par épuisement de ressources

Les plus populaires.

- *SYN flood* : envoi de requêtes répétées de connexion.
  - Le système cible doit mémoriser le no. de séquence initial pour pouvoir bien établir la connexion.
  - Les nos. de séquence sont mémorisés dans une queue de connexion.
  - Si trop de nos. de séquence arrivent, cela va provoquer une saturation de la queue de connexion...
  - ... et donc bloquer le serveur, qui ne pourra plus répondre à des nouvelles démandes, même légitimes, d'ouverture de connexion (nouveau paquet SYN).
  - Pour plus de furtivité (duper les pare-feux ou IDS), les paquets peuvent être envoyés avec des adresses source déguisées.

Défenses contre des SYN floods

- Augmenter la taille de la queue de connexion de son serveur.
- SYN cookies : modifier légèrement le comportement de la pile TCP/IP, en calculant, sur la base de l'adresse IP de la source et du destinataire, et d'un no. secret, un ISN_B.
- La machine source de la demande de connexion fera le même calcul, à partir de son no. secret.
- Si la machine cible reçoit la bonne réponse dans le paquet final ACK, la connexion sera ouverte.
- Patch Linux : `tcp_syncookies` à exécuter au boot.
- "Façonneur" de traffic (angl. *traffic shaper*) sur les proxies : limitateurs de no. de connexions qu'une machine peut recevoi d'une connexion réseau.

Attaques à distance par épuisement de ressources

- Attaques *Smurf* :
  - Broadcast vers une adresse IP, avec adresse de réponse déguisée sur la cible de l'attaque.
  - Les machines intermédiaires font l'*amplificateur*.
  - Différents attaques, basées sur ICMP, UDP (port 7, UDP echo service).
- Défenses :
  - Du coté cible : patches correctifs.
  - Du coté amplificateur : stopper les réponses directionnées de broadcast au router vers l'extérieur.

Attaques à distance par épuisement de ressources

- Attaques distribuées (DDoS).
  - Prise de contrôle d'un large no. de machines "zombie" (serveurs vulnerables des universités, ISP, companies de toute taille, même des machines domestiques connectées en permanence à l'Internet).
  - Commande à distance par l'intermédiaire de "clients", qui les rélaient aux zombies.
  - Bcp de "maliciels" dédiés aux attaques DDoS : TFN2K, Blitznet, Shaft...
- Défenses :
  - Ne pas se faire transformer en zombie : outils de détection (*Find DDOS*, US NIPC).
  - Garder une *bande passante* suffisament large (Internet).
  - Mais sourtout détection rapide et réaction rapide aux attaques (en filtrant les mauvais paquets).

Récupérer des informations sensibles

- Récuperation d'informations sensibles disponibles en ligne ou par "ingénierie sociale".
- Balayage de ports/protocôles : l'attaquant peut dresser une liste de services disponibles sur la victime.
  - Peut aussi découvrir les caractéristiques des protocôles
  - ... et donc cibler ses attaques !
- Utilisation des protocôles pour faire transiter de l'information cachée.
  - Covert channels.
  - Utilisation des paquets ICMP Host Unreachable ou Echo Request.
  - Utilisation du protocôle HTTP.
- Fuite d'information à l'interieur même d'un système
  - Utiliser des fichiers partagés pour propager l'information.
  - Transmettre un bit d'information par épuisement de ressources.

Balayage par paquets TCP

- Balayage “poli” – connexion complète.
  - Si port ouvert, envoi SYN-ACK, sinon RST ou ICMP Port Unreachable.
- Balayage “furtif” – scanning de type SYN
  - Envoi seulement de paquets SYN.
  - Si réponse SYN-ACK, renvoi de paquets RST.
  - Sinon, reception de paquets RST ou ICMP Port Unreachable.
- Violations de la spécification :
  - Scanning de type TCP FIN, sans avoir de connexion ouverte.
    - Pour certaines specs TCP, si port destinataire fermé, renvoi de RST, sinon, pas de réponse.
  - “Sapin de Noël” : paquet FIN, URG, PSH.
  - `Null` scanning : paquet sans bit de code.
    - Le même comportement attendu que pour le TCP FIN.
  - Scanning de type TCP ACK : duper les pare-feux.
    - Si pare-feu accepte connexions initiées de l’interieur.
    - Port ouvert – renvoi de TCP RST, sinon ICMP Port Unreachable.