Python et Unix (III)

Yann Thierry-Mieg

Laboratoire d'informatique de Paris 6

EPSI

Introduction à la sécurité

1. Vers, virus, trojans
2. Attaques classiques
3. Utilitaires
   - crypt, ssh, pgp

I : Préoccupations en Sécurité

Sécurité : responsabilités

- S'assurer qu'une donnée n'est accessible qu'a ceux qui en ont l'autorisation
- S'assurer que les utilisateurs ne font pas des choses interdites (applications ...)
- Protéger les données contre la corruption et la perte

Terminologie

- Authentification :
  - s'assurer qu'une personne est bien qui elle prétends
- Autorisation :
  - s'assurer qu'une personne a bien l'autorisation de faire ce qu'elle demande
- Intégrité
  - S'assurer qu'une donnée n'est pas corrompue

Approche extrême : Le point de vue de l'autruche

- Eliminer toutes les connexions entrantes et sortantes avec l'exterieur
- Uniquement autoriser l'accès depuis un terminal directement branché sur la machine
- Placer machine et terminal dans une piece isolée electriquement (cage de Faraday)
- Mettre un garde devant la porte
- Utilisation légitime difficile...

Politique de sécurité = approche globale

- Besoin de classifier :
  - le niveau de risque
  - le niveau de protection à atteindre
- Etablir la politique de sécurité en conséquence

Niveaux de sécurité vs. autres critères

- Confidentialité :
  - Empêcher une information d'être lue ou copiée par quelqu'un qui n'ait pas été explicitement autorisé à le faire
- Intégrité des données :
  - Empêcher la modification/suppression des données ou des programmes sans la permission du *propriétaire* de l'information
- Disponibilité :
  - Protéger vos services pour qu'ils ne puissent être dégradés voire crashés sans authorisation

Niveaux de sécurité vs. autres critères (2)

- Consistance :
  - S'assurer que le système se comporte comme l'attendent les utilisateurs légitimes (i.e. patch...)
- Contrôle :
  - Réguler les accès au système, savoir quand il y a une pénétration du système, être capable d'identifier le trou de sécurité mis en cause
- Audit :
  - Etre capable de tracer les activités des utilisateurs (légitimes ou non). Etre capable en cas d'erreur ou d'attaque de déterminer qui a fait quoi dans le système, et quelles données ont été modifiées

Niveaux de sécurité vs. autres critères (3)

- Dans une banque :
  - Intégrité et Audit sont prioritaires, suivis de confidentialité et disponibilité
- Dans une activité secret-défense :
  - La confidentialité prime sur ***tout*** autre aspect. La disponibilité du système est la moindre préoccupation
- Dans une fac/école :
  - Intégrité et disponibilité sont souvent prioritaires. La priorité est que les étudiants puissent travailler, pas de tracer leurs activités

II : Vers et autres virus

Vers et trojans

- Cheval de troie (Trojan) :
  - Programme qui compromet la sécurité en prétendant en être un autre
- Virus :
  - code malicieux qui modifie (se copie) dans des programmes sains pour se dupliquer
- Ver (Worm):
  - Code malicieux capable de s'auto-diffuser à travers un réseau

Social Engineering

- Mentir pour abuser la confiance des utilisateurs (i.e. Kevin Mitnick)
- Une des façons les plus faciles de compromettre la sécurité
- Appels téléphoniques bidons, difgfusion de mails, ballade dans les couloirs..
- Difficile à contrer :
  - Eduquer les gens qui ont accès aux informations sensibles
  - Limiter l'accès aux données sensibles

Attaques par base de connaissance

- Attaquer les ignorants :
  - password par défaut activés
  - Bugs de sécurités diffusés mais non patchés sur le système cible
  - War-dial : scanner des IP/port, à la recherche de services activés
  - Oublier de gérer les permissions de fichiers
    - qui va s'embêter à aller voir ce fichier ?
    - réponse : un script !!

Snoop, sniff

- En écoutant passivement on peut récuperer toutes sortes d'informations : passwords, ...
- Très (TRES) facile à faire
  - TCP/IP n'est pas crypté
  - Un paquet travers toute une série de machines
  - Facilité d'obtention d'outils d'analyse du traffic
- i.e. : Ethereal, http://www.insecure.org/tools.html

Spoofing

- Un attaquant crée un contexte pour tromper sa victime
  - i.e. les faux DAB
- Mentir sur les adresses d'origine et/ou le uid de l'utilisateur sur rsh/rcp/rlogin:
  - permet de se logger si .rhosts le permet
- Spoofing de page web :
  - Pages contenant des erreurs permettant d'accèder à une machine
  - Prétendre d'être une page officielle et demander pass/login
  - CSS : cross site scripting, utiliser des champs mal protégés pour insérer du code dans la page visée

Denial of Service : DoS

- Objectif n'est pas d'accéder au système, mais d'empêcher l'accès par les utilisateurs légitimes
  - i.e. amazon
- Exemple : ping en faussant les adresses émetteur (cf. hping2)
- Exemple 2 : SYN/ACK = début de connection TCP mais ne pas répondre à l'ACK serveur, qui mobilise des ressources jusqu'au timeout (3 minutes)
- DDOS : DoS distribués depuis plusieurs (1000) machines, beacoup plus difficile à empêcher

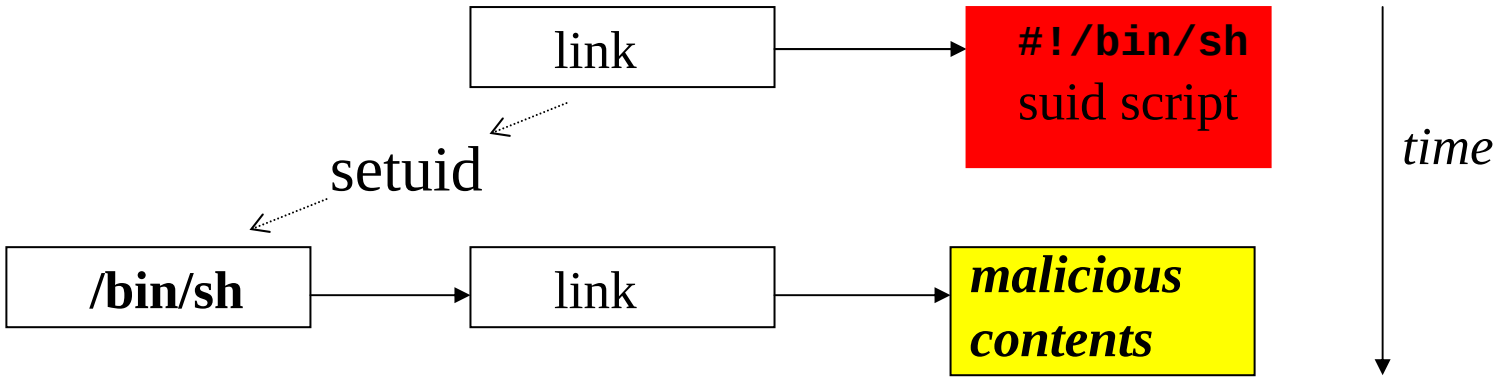
III :Passwords et protection

Scripts et bit s

- Scripts en suid/sgid peuvent être agréable pour les tâches administratives
  - i.e tuer les applications plantées (netscape) des autres utilisateurs sans être root
- Uniquement des scripts aussi petits que possible
- Attention aux vulnérabilités :
  - Ne mettez pas ./ ou tout autre répertoire relatif dans votre $PATH
  - N'utilisez pas ***eval*** dans vos scripts
  - Attention aux fichiers temporaires

Une vulnérabilité de script en bit s

- #! invoque la première ligne du script avec comme argument le nom du script
- Danger : créer un lien sur un script suid, puis entre l'invocation du script et l'exécution du programme spécifié par #! on change le lien pour pointer sur un autre script



crypt : password unix

- Les password sont cryptés de façon irréversible par crypt (man 3)
- Stockés dans /etc/passwd ou /etc/password.shadow ou sur le NIS (ypcat)
- Utilise un *sel* pour augmenter la complexité :
  - deux premiers caractères du password crypté
  - **s9**dl30c3LPqV
  - Rend plus difficile les attques au dictionnaire (complexité * 4096)

Vulnérabilité de crypt

- Initialement conçu pour être couteux en temps de calcul
- Avec les puissances des machines, l'attaque en force brute reste possible
- Améliorer à l'aide de dictionnaire (cf. TP)
- Crypter tous les mots d'un dico et comparer les valeurs cryptées aux entrées de /etc/passwd

Eviter ces vulnérabilités

- Forcer à l'introduction d'un password:
  - un essai d'attaque au dictionnaire
  - imposer des longueurs minimales
- utiliser /etc/password.shadow :
  - fichier qui contient les password cryptés
  - interdit en lecture aux utilisateurs
  - accès réservé aux scripts d'authentification légitimes
- Essayer périodiquement de cracker son système
- Dates d'expiration :
  - controversé, trop de changements => syndrome post-it

Protéger la confidentialité : crypt

- Algorithme de cryptage inspiré de Enigma
  - f(clair) = codé
  - f(codé) = clair
- Filtre sur stdin/stdout :
  - crypt < clair.txt > crypte.txt
  - prompte pour une phrase pass
- Certains éditeurs (vi,emacs) capables de travailler sur de tels fichiers
  - vi –x crypte.txt
- Mauvais niveau de sécurité
  - cbw : crypt breaker's workbench casse la protection

Cryptage avec clé publique

- Cryptage normal (i.e. DES) :
  - Fonction d'encryptage : E(clé, texte)
  - Fonction de décryptage : D(clé,texte)
  - D(clé, E(clé,texte)) = texte
  - La clé est privée
- Clé publique :
  - clé_pub = f(clé)
  - E(clé_pub,texte) = E (clé,texte)

    MAIS
  - D(clé_pub,texte_crypté) != D(clé,texte_crypté)
  - La clé publique est rendue publique, la clé est privée

Exemples utilisant une clé publique

- RSA :
  - Système de Rivest, Shamir, Adleman
  - Complexité basée sur la difficulté de factoriser en nombres premiers de grands nombres
- PGP :
  - Pretty Good Privacy
  - Similaire à RSA, mais mélange d'autres approches
  - Sur un brevet différent de RSA => gratuit/libre

Combien de bits pour la clé ?

- Théoriquement on peut essayer toutes les clés

•**Taille clé (bits)**	•**Temps (1s/test)**	•**Temps (1s/10^6test)**
•32	•35.8 mins	•2.15 msec
•40	•6.4 jours	•550 msec
•56	•1140 ans	•10.0 heures
•64	•~500000 ans	•107 jours
•128	•5 x 10^{24}ans	•5 x 10^{18} ans

Sécurité du réseau

- Il est facile de d'écouter (sniffer) les paquets
- Très dangereux en telnet car le password passe en clair (ou ftp, pop, ...)
- Un intermédiaire quelconque ou du réseau local s'il n'est pas switché (et encore) peut récuperer les password

Première approche

- Commandes rsh, rlogin, rcp introduites en BSD
- But :éviter de faire transiter les mots de passe sur le réseau
- Authentifier en vérifiant :
  - si la machine origine est dans /etc/hosts.equiv
  - si le port d'origine est privilégié
  - si utilisateur et machine sont spécifiés dans /.rhosts
- Problèmes :
  - permissions de fichiers
  - propagation d'un problème de sécurité au réseau entier
  - spoofing d'identité reste possible(NFS et le compte root)

Secure Sockets

- SSL : Secure Socket Layer
- Se comporte comme une socket TCP/IP normale
- A la connexion :
  - client et serveur s'echangent leur clé publiques
  - chaque extrémité crypt le flux sortant avec la clé publique de l'autre, et décrypte le flux entrant avec sa clé privée
- Certificats :
  - Permettent d'assurer que la clé publique appartient bien à qui l'on croit (man-in-the-middle) (.ssh/known_hosts)

Protocoles utilisant SSL

- ssh :
  - telnet en ssl
  - inclut scp et sftp en remplacement de rcp et ftp
- https : secure http
  - utilisé sur sites web sécurisés (CB ...)

Bibliographie

- **Linux Network Administrator's guide,** Olaf Kirch and Terry Dawson, *O'reilly*
- **Running Linux, 4th Edition,** Dalheimer, Dawson, Kaufman, Welsh, *O'Reilly*
- **Practical UNIX and Internet security,** Simson Garfinkel & Gene Spafford, 2nd ed. *O'reilly*
- **Understanding the Linux Kernel,** Daniel P. Bovet, Marco Cesati, *O'Reilly*
- ICTP: invent yourself, centre de formation en ligne
- Système : Noyau, Pierre Sens, Supports de maîtrise d'informatique Paris 6

Readline pour l'autocompletion

- Fournit également tout les raccourcis usuels du shell unix (C-a,C-e,C-_ ...)

```
>>> import rlcompleter
>>> import readline
>>> readline.parse_and_bind('tab: complete')
>>> r                                                    #tab
raise       raw_input   reduce      repr        rlcompleter
range       readline    reload      return      round 1
>>> re                                                   #tab
readline  reduce    reload    repr      return
```