CRÉER UNE YUBIKEY SANS YUBIKEY

Les modules de sécurité PAM / SELinux / Udev / SSL en C(++)

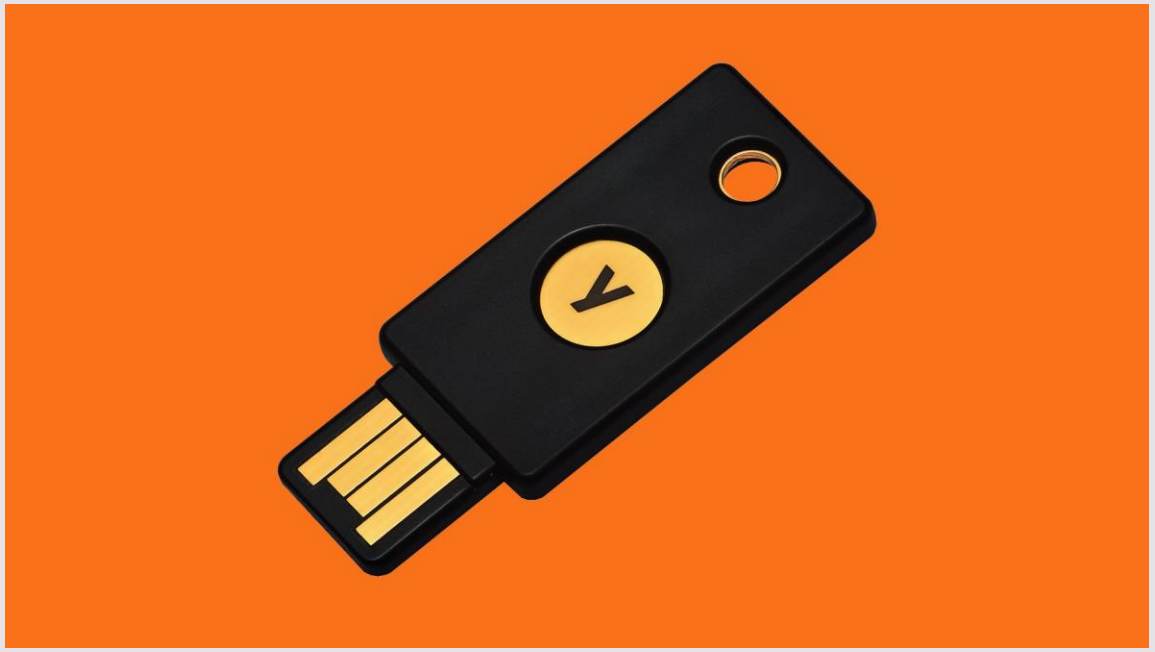
SOMMAIRE

- Présentation du projet & objectifs
- Modules de sécurité PAM
- Utiliser et implémenter SSL
- Règles Udev
- L'isolation des processus par SELinux

CHECK THIS OUT!!

`github.com/AshleyTheNeko/auth_key_workshop`

PRÉSENTATION DU PROJET & OBJECTIFS



- Une clé d'authentification, remplaçant/ignorant un mot de passe
- Utilisable pour sudo, screensaver, desktop manager & autres
- Sécurisé, impossible à reproduire
- Beaucoup de notions techniques utiles en sysadmin

MODULES DE SÉCURITÉ PAM

(P)luggable (A)uthentication (M)odules

- Créer un système d'authentification et de maintien de session modulaire, géré par des programmes tiers
- API de PAM en C/C++, utilisant des objets .so
- Fichiers de configurations uniques pour chaque programme

LES FICHIERS DE CONFIGURATION PAM

- Trouvables dans `"/etc/pam.d/"`
- Définissent les règles à appliquer lors d'une demande des applications
- Chaque fichier correspond à la configuration d'un programme
- Une mauvaise modification entraîntera une faille de sécurité ou un login impossible !!

FONCTIONNEMENT DES FICHIERS DE CONFIGURATION

Quatre types de modules :

- `Account`: vérifier la disponibilité du compte
- `Password`: changer/approuver le jeton d'authentification
- `Session`: tâches à effectuer à l'authentification/déconnection
- `Auth`: validation de l'ouverture d'une session (that's what we do!!)

Quatre règles de validation:

- `Required`: indispensable, si échoué, toute la chaîne est lue, mais l'authentification est refusée
- `Requisite`: indispensable (plus fort), si échoué, toute la chaîne est arrêtée
- `Sufficient`: valide instantanément la requête sans lire le reste de la chaîne
- `Optional`: lu mais complètement ignoré dans la validation

Créer un module PAM

- Pour notre cas, il s'agit d'un module d'authentification

- Les fonctions PAM sont disponibles en C en ajoutant :

```
#define PAM_SM_AUTH
#include <security/_pam_macros.h>
#include <security/pam_modules.h>
```

- La fonction à implémenter est :

```
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags,
                                   int argc, const char **argv)
```

ÉCRIRE LES FONCTIONS PAM

- Un test à effectuer

  `pam_get_user(pamh, &username, NULL) == PAM_SUCCESS`

  afin de vérifier si l'utilisateur existe bien, en cas d'erreur, retourner `PAM_AUTHINFO_UNAVAIL`

- Pour valider l'authentification : retourner `PAM_SUCCESS`

- Pour refuser l'authentification : retourner `PAM_AUTH_ERR`

Compilation du module PAM

```
gcc pam_module.c -Wall -Wextra -fpic -c

ld -shared -lpam -lcrypto -lssl
pam_module.o -o pam_my_module.so
```

AJOUTER VOTRE MODULE PAM

- Déplacer le fichier .so dans `"/lib64/security/"`

- Exécuter la commande suivante (explications plus tard !) :

```
sudo restorecon -v -R /lib64/security
```

- Ajouter le nouveau module dans les fichiers de configuration concernés
  - Exemple pour sudo :

```
auth    sufficient    pam_secure_key.so
```

UTILISER ET IMPLÉMENTER SSL

SSL, QU'EST-CE QUE C'EST ?

- Utilisé dans le chiffrement de donnés au cours d'un échange entre serveur et client
- Dans ce cas, utilisé en tant que certificat
- Deux clés, une clé privée dans la clé usb, une clé publique sur l'ordinateur
- Pourquoi ce choix ? Rapide, et réutilisable pour d'autres projets

Créer (et graver) la clé SSL

- Pour générer la paire de clés, utiliser la commande

  ```
  openssl genrsa -out rsa 4096 && openssl rsa -in rsa -outform PEM -pubout -out rsa.pub
  ```

- Pour ajouter la clé privée RSA dans la clé usb, vous pouvez soit l'ajouter dans un fichier sur la clé usb, soit la graver directement sur la clé
  - Pour graver la clé, vous pouvez utiliser

    ```
    sudo dd if=rsa of=/dev/[clé]
    ```

    Attention avec `dd` !! Cette commande peut effacer un disque **ENTIER**, vérifiez bien où vous l'utilisez.

IMPLÉMENTER LA CLÉ SSL DANS LE MODULE

- Pour vérifier si les clés correspondent, dans notre cas, il suffit de vérifier si les paramètres N des deux clés correspondent

- Pour cela, utiliser `"BN_cmp(RSA_get0_n(pubkey), RSA_get0_n(privkey)"` en appelant `"PEM_read_bio_RSAPrivateKey"` pour charger la clé privée lue avec `read`, et `"PEM_read_RSA_PUBKEY"` en C

RÈGLES UDEV

QU'EST-CE QUE UDEV ?

- Le sous système linux de gestion des appareils externes !
- Permet de changer les permissions d'un appareil, ou de le monter dès son branchement
- Fichiers de configuration personnalisés dans `"/etc/udev/rules.d/"`

CONFIGURER UNE RÈGLE UDEV

- Les règles Udev ont une syntaxe :
  - En premier une comparaison, permettant de déterminer sur quel appareil l'action se déclanche
  - En deuxième des actions à effectuer

- Exemple de règles :
  - `KERNEL=="sd[a-d]", MODE="0664"`
  - `BUS=="usb", ATTR{manufacturer}=="OLYMPUS", ATTR{product}=="X250,D560Z,C350Z", MODE="0664"`
  - `KERNEL=="sd[a-d]", RUN+="/bin/mount /dev/%k /mnt/key_%k"`

L'ISOLATION DES PROCESSUS PAR SELINUX

(SE)CURITY-(E)NHANCED (LINUX)

- Une police d'interaction entre processus et fichiers
- Permet de définir les actions autorisées d'un programme
- Chaque utilisateur, fichier, programme possède un contexte SELinux
- Le contexte contient :
  - Une identité `(_u)` : défini quel utilisateur possède le fichier, et quels rôles le fichier peut avoir
  - Un rôle `(_r)` : défini dans quels domaines le fichier peut être
  - Un domaine `(_t)` : permet de définir des règles d'accès (par ex: un domaine `xdm_t`, attribué au desktop manager) pourra accéder à un domaine `xauth_home_t`, contenant par exemple `.Xauthority`, utile pour démarrer une session X)

TRAVAILLER AVEC SELINUX

- Obtenir le contexte d'un fichier : `ls -Z [fichier]`
- Obtenir le contexte d'un programme : `ps -eZ | grep [programme]`
- Autoriser une action entre deux domaines SELinux :
  - Option complexe mais sécurisée : Ajouter une règle dans la police SELinux concernant un domaine spécifique (Ne sera pas abordé dans ce workshop)
  - Option simple : "`sudo semanage permissive -a xdm_t`"