Everybody be Cool, This is a Robbery!

Jean-Baptiste Bédrune, Gabriel Campana

Hong Kong - New York - Paris – San Francisco - Vierzon

Plan

- Qu’est ce qu’un HSM ?
- Présentation du HSM étudié
- Brève introduction à PKCS #11
- Développement d’outils pour la recherche de vulnérabilités
- Recherche de vulnérabilités et exploitation
- Compromission permanente

Introduction

Qu’est-ce qu’un HSM ?

Ledger

- Enclave de sécurité qui protège et manipule des données sensibles
  - Effectue des calculs cryptographiques
  - Génère des clés
  - Les clés ne quittent jamais l’enclave
- Périphérique matériel :
  - Appliance réseau ou carte PCI
  - Un ou plusieurs cryptoprocesseurs
  - Contre-mesures en cas d’altération

Exemples d’utilisation

Ledger

- IGC :
  - Génération et stockage des clés privées des CA, signature des certificats
  - Obligatoire pour toutes les CA (*CA/Browser Forum Baseline*)
- Banques : vérification du PIN, validation des transactions, personnalisation des cartes de paiement
- Télécommunications : personnalisation des cartes SIM, authentification
- DNSSEC : stockage des clés racine (FIPS 140-2 niveau 4)
- Services Cloud : chiffrement / déchiffrement des données des clients
- HSM-as-a-Service : Google, Microsoft, Amazon, etc.

Combien ça coûte ?

Ledger

- Peu de fabricants, pas d'informations publiques sur les parts de marché
- Pas de prix publics, nombreux modèles pour chaque fabricant
- D'après Hackable Security Modules (Recon 2017) :
  - Fabricant X, Modèle A : 29 500 $
  - Fabricant Y, Modèle B : 9 500 $
  - Fabricant Z, Modèle C : 15 000 $

Caractéristiques du HSM étudié

Photo d’illustration. Source : Google Images

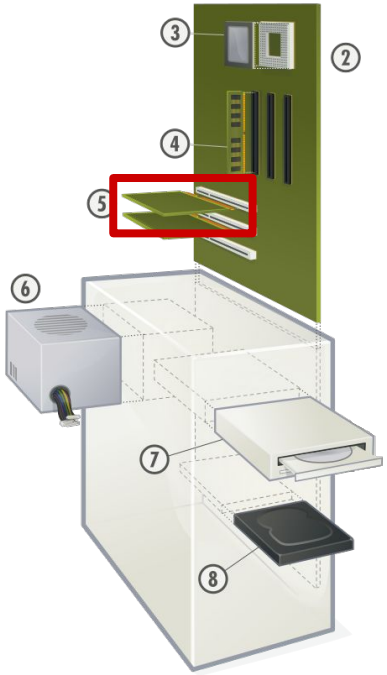
- Carte PCIe (existe aussi sous forme d’appliance réseau)
- Certification FIPS 140-2 niveau 3
- Composants cachés sous une couche d’ époxy
- Ports série et USB permettant de connecter un éventuel lecteur de carte à puces
- Contrôleur Ethernet, sans connecteur

FIPS 140-2: Exigences de sécurité pour les modules cryptographiques

- Niveau de sécurité établie par le gouvernement des Etats-Unis
- Niveau 1:
  - Au moins un algorithme approuvé ou une fonction de sécurité approuvée
  - Pas de mécanisme de sécurité physique spécifique
  - Exemple : carte de chiffrement pour PC
- Niveau 2 : mécanisme de sécurité physique. Preuve d'altération (enrobage, sceaux)
- Niveau 3 : détection et remédiation aux tentatives d'accès physique, à l'utilisation ou à la modification du module cryptographique
- Niveau 4 : plus haut niveau, conçu pour les environnements non protégés physiquement

Appliance (hôte)

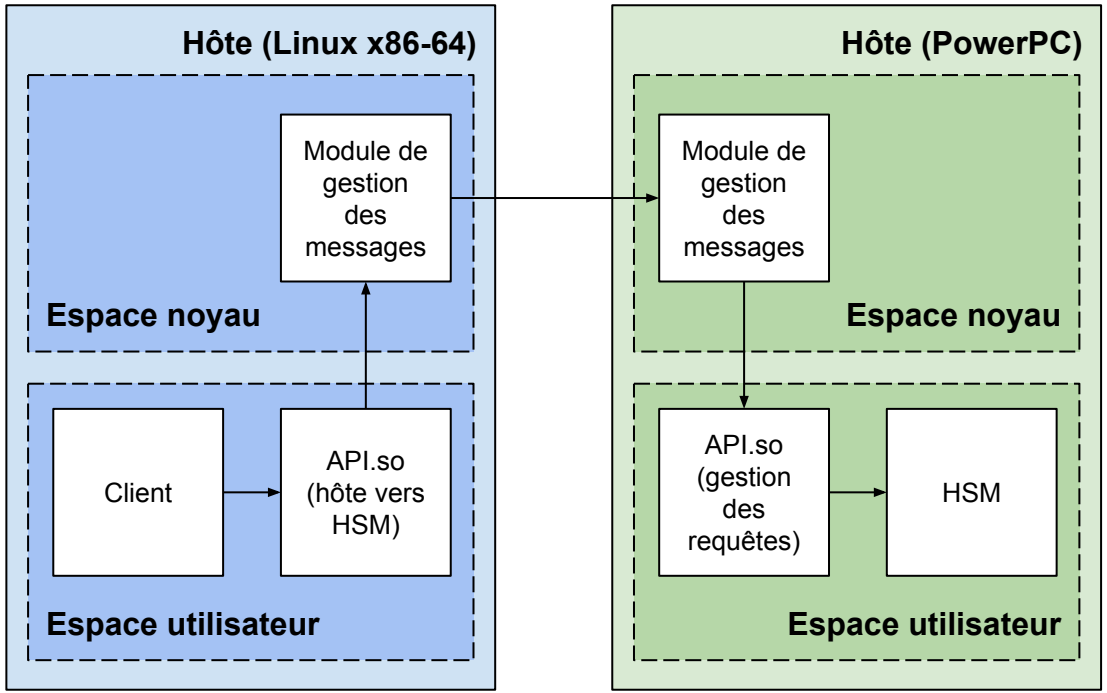
- Serveur Linux classique
  - Processeur : Intel E5500
  - RAM : 4 Go
  - Disque dur : 500 Go
- Modules noyau
- Outils graphiques et en ligne de commande
- SDK

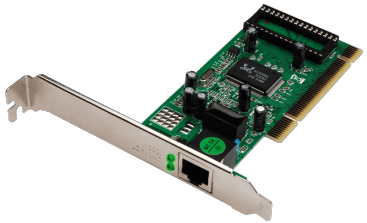


Firmware

- CD-ROM fourni par le fabricant
- Logiciels graphiques et en ligne de commande
- Exemples d’utilisation de l’API PKCS #11 en C
- Documentation pour les développeurs et les administrateurs
- Mise à jour du firmware : en clair, signé, Linux 2.26.8.8 pour PowerPC

Communication : DRAM partagée





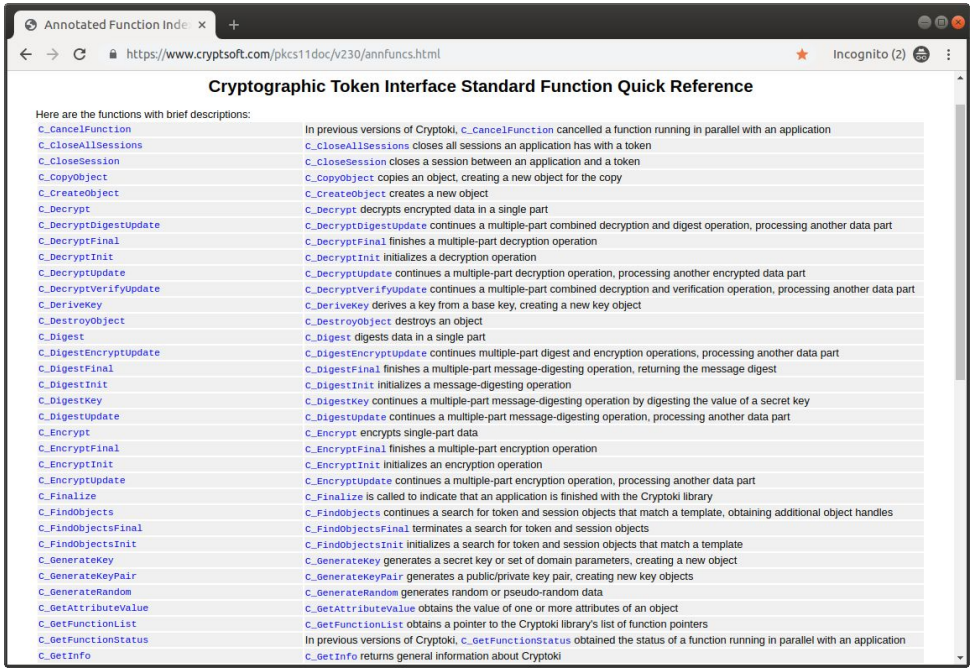
PKCS #11

PKCS #11 : Introduction

Ledger

- Interface générique pour communiquer avec un périphérique cryptographique
  - Carte à puce
  - HSM, etc.
- API portable : Cryptographic Token Interface (Cryptoki)
  - Création et manipulation des objets cryptographiques
  - Opérations avec ces objets (chiffrement, déchiffrement, signature, etc.)
  - Gestion des sessions

Cryptographic Token Interface



Cryptographic Token Interface Standard Function Quick Reference

Here are the functions with brief descriptions:

C_CancelFunction	In previous versions of Cryptoki, C_CancelFunction cancelled a function running in parallel with an application
C_CloseAllSessions	C_CloseAllSessions closes all sessions an application has with a token
C_CloseSession	C_CloseSession closes a session between an application and a token
C_CopyObject	C_CopyObject copies an object, creating a new object for the copy
C_CreateObject	C_CreateObject creates a new object
C_Decrypt	C_Decrypt decrypts encrypted data in a single part
C_DecryptDigestUpdate	C_DecryptDigestUpdate continues a multiple-part combined decryption and digest operation, processing another data part
C_DecryptFinal	C_DecryptFinal finishes a multiple-part decryption operation
C_DecryptInit	C_DecryptInit initializes a decryption operation
C_DecryptUpdate	C_DecryptUpdate continues a multiple-part decryption operation, processing another encrypted data part
C_DecryptVerifyUpdate	C_DecryptVerifyUpdate continues a multiple-part combined decryption and verification operation, processing another data part
C_DeriveKey	C_DeriveKey derives a key from a base key, creating a new key object
C_DestroyObject	C_DestroyObject destroys an object
C_Digest	C_Digest digests data in a single part
C_DigestEncryptUpdate	C_DigestEncryptUpdate continues multiple-part digest and encryption operations, processing another data part
C_DigestFinal	C_DigestFinal finishes a multiple-part message-digesting operation, returning the message digest
C_DigestInit	C_DigestInit initializes a message-digesting operation
C_DigestKey	C_DigestKey continues a multiple-part message-digesting operation by digesting the value of a secret key
C_DigestUpdate	C_DigestUpdate continues a multiple-part message-digesting operation, processing another data part
C_Encrypt	C_Encrypt encrypts single-part data
C_EncryptFinal	C_EncryptFinal finishes a multiple-part encryption operation
C_EncryptInit	C_EncryptInit initializes an encryption operation
C_EncryptUpdate	C_EncryptUpdate continues a multiple-part encryption operation, processing another data part
C_Finalize	C_Finalize is called to indicate that an application is finished with the Cryptoki library
C_FindObjects	C_FindObjects continues a search for token and session objects that match a template, obtaining additional object handles
C_FindObjectsFinal	C_FindObjectsFinal terminates a search for token and session objects
C_FindObjectsInit	C_FindObjectsInit initializes a search for token and session objects that match a template
C_GenerateKey	C_GenerateKey generates a secret key or set of domain parameters, creating a new object
C_GenerateKeyPair	C_GenerateKeyPair generates a public/private key pair, creating new key objects
C_GenerateRandom	C_GenerateRandom generates random or pseudo-random data
C_GetAttributeValue	C_GetAttributeValue obtains the value of one or more attributes of an object
C_GetFunctionList	C_GetFunctionList obtains a pointer to the Cryptoki library's list of function pointers
C_GetFunctionStatus	In previous versions of Cryptoki, C_GetFunctionStatus obtained the status of a function running in parallel with an application
C_GetInfo	C_GetInfo returns general information about Cryptoki

- Peu de fonctions (~70)
- Mais plus de 250 mécanismes (standards et propriétaires)

PKCS #11 : Mécanismes

Ledger

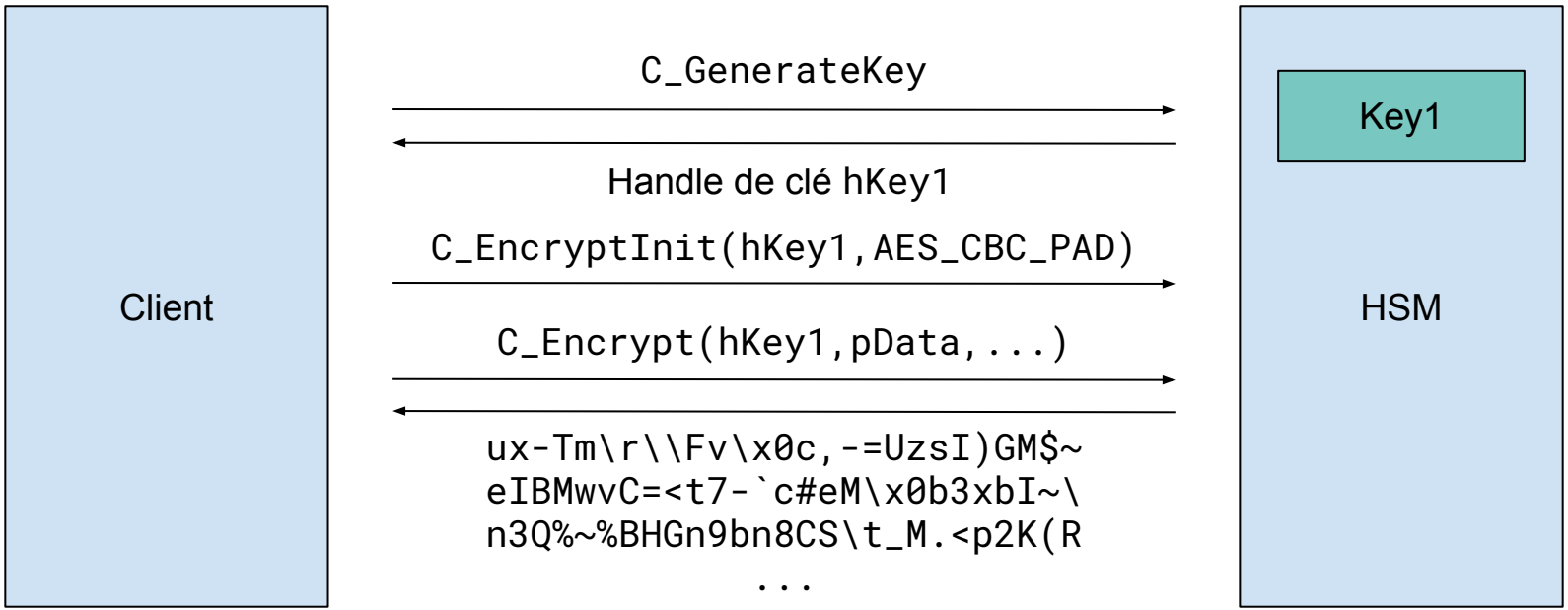
- Décrit une opération cryptographique.
- Mécanismes pour le chiffrement, le déchiffrement, le hachage, le wrapping, etc.
- Dépend du périphérique. HSM : beaucoup de mécanismes.
  - `CKM_SHA512_HMAC`
  - `CKM_RSA_PKCS_KEY_PAIR_GEN`
  - `CKM_WRAPKEY_AES_CBC`
  - `CKM_AES_GCM`
  - Mécanismes pour les télécoms, la banque…
- Certains mécanismes requièrent des paramètres : IV, sel, etc.

PKCS #11 : Objets

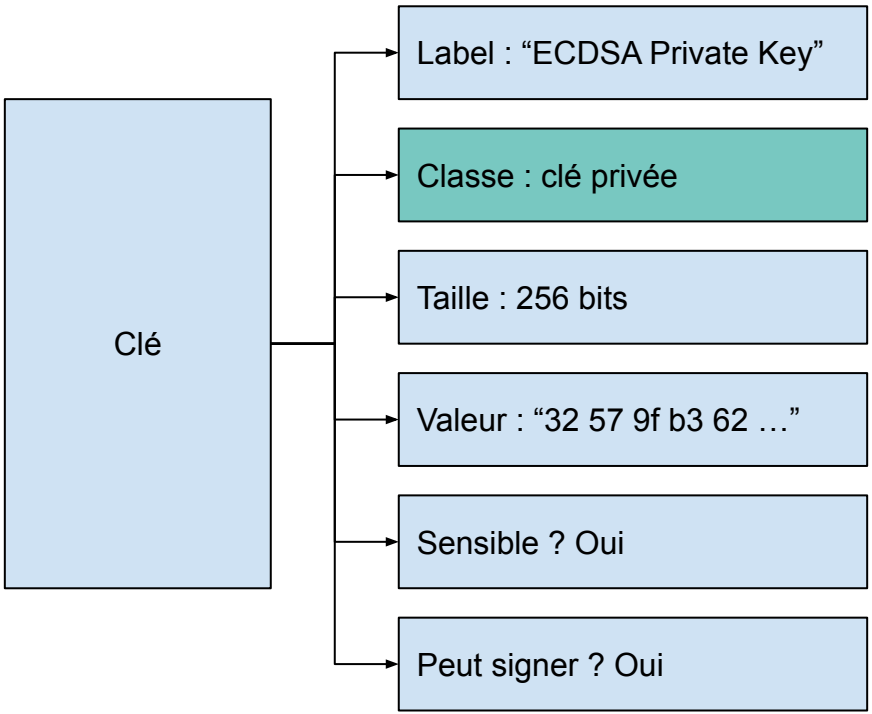
Ledger

- 3 types
  - Clés : secrètes, publiques, privées
  - Certificats
  - Données : paramètres de domaine DSA et ECDSA, etc.
- Cryptoki manipule des objets au travers de leurs handles

PKCS #11 : Objets

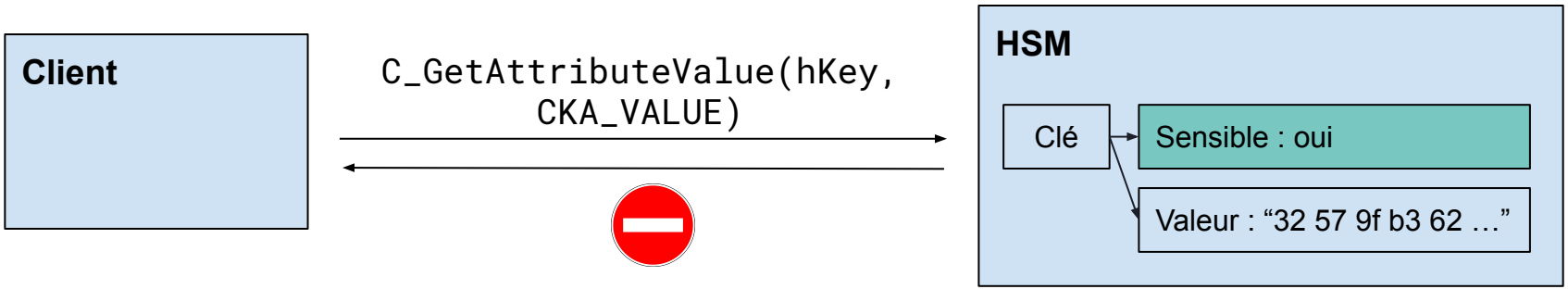


PKCS #11 : Attributs d'objets

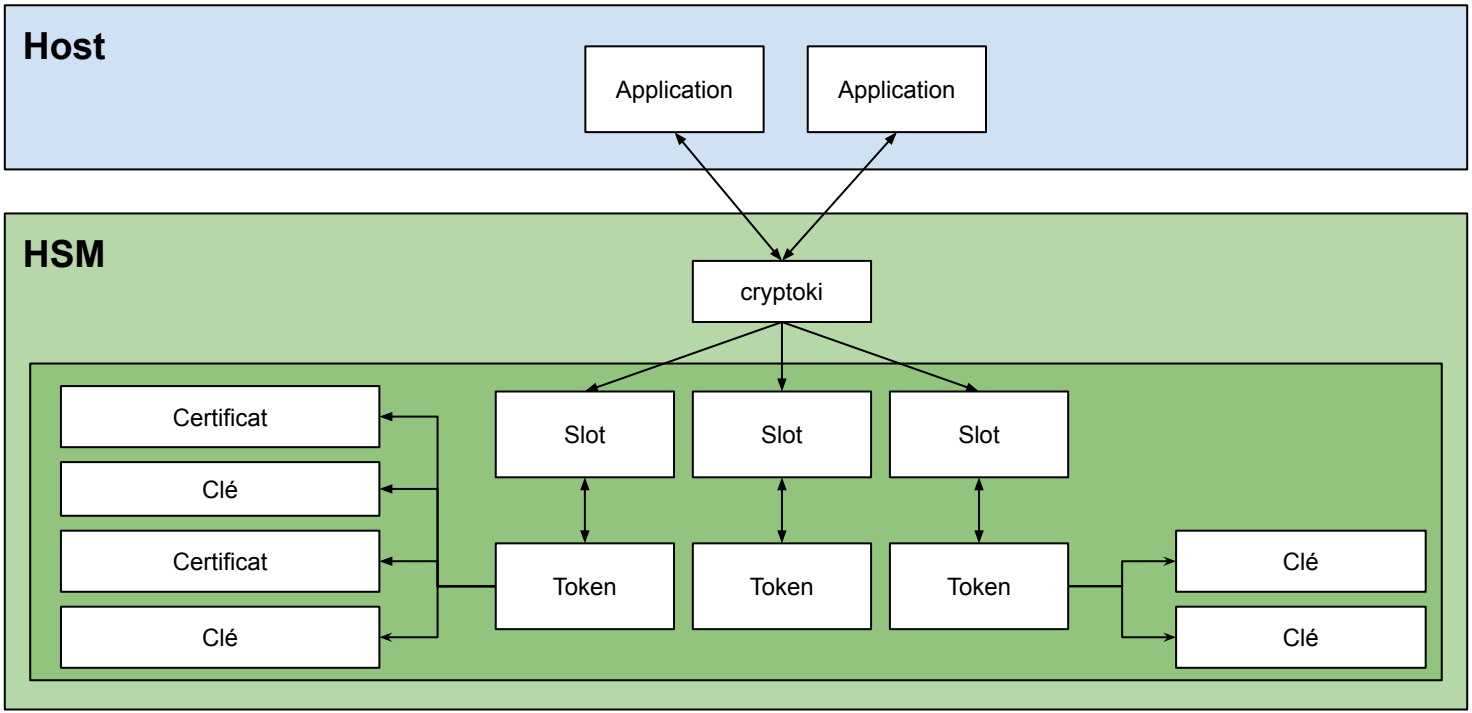


PKCS #11 : Attributs pour la sécurité

- Sensible : la valeur ne peut pas être extraite en clair, elle doit être wrappée
- Non extractible : la valeur ne peut pas être exportée



Slots et tokens



PKCS #11 : Rôles

Ledger

- Deux rôles :
  - Utilisateurs : peuvent accéder aux objets privés d'un token
  - Administrateurs : peuvent gérer les comptes
- Menaces :
  - Un attaquant non authentifié accède à des objets privés
  - Un attaquant extrait des clés marquées comme non extractibles
  - Un attaquant avec des droits utilisateur élève ses privilèges

Recherche de vulnérabilités et exploitation

Modèle d’attaque

- L’attaquant a le contrôle de la machine hôte
  - Employé malveillant
  - Accès physique à la machine dans le datacenter
  - Compte administrateur compromis
  - Vulnérabilités logicielles sur l’hôte (ex : iDRAC)
  - etc.

Outillage

Modules personnalisés

Ledger

- Nouvelles fonctionnalités grâce à des *modules* personnalisés
- Exemple d'utilisation :
  - Hook de fonctions PKCS #11
  - Ajout de nouveaux types de messages
- Nécessite les privilèges administrateur pour être chargé
- Pas une vulnérabilité
- Fonctionnement :
  - Le SDK et la toolchain génèrent des binaires ELF PowerPC
  - Les modules sont chargés dans le processus principal grâce à `dlopen()`
  - Pas de `libc`

Shell

Ledger

```
user@host:~$ ./module-shell --init
[*] uploading busybox-powerpc to /sbin/busybox
[*] creating symlinks (might take a few seconds)

user@host:~$ ./module-shell id
uid=0 gid=0

user@host:~$ ./module-shell ps fauxwww
PID   USER     TIME   COMMAND
    1 0          0:00 /init
    2 0          0:00 [kthreadd]
…
 1086 0          0:00 /sbin/busybox ps fauxwww
```

Débogueur

- Le processus principal gère les communications
- Les fonctions du SDK ne peuvent pas être utilisées
- Recherche de canaux de communication *auxiliaires* (ex : mémoire partagée)
- gdb sur l'hôte, `gdbserver` sur le HSM
- Problématiques additionnelles :
  - Le processus principal est monitoré avec *ptrace*
  - Le HSM est redémarré lorsqu'un crash a lieu

Informations récupérées

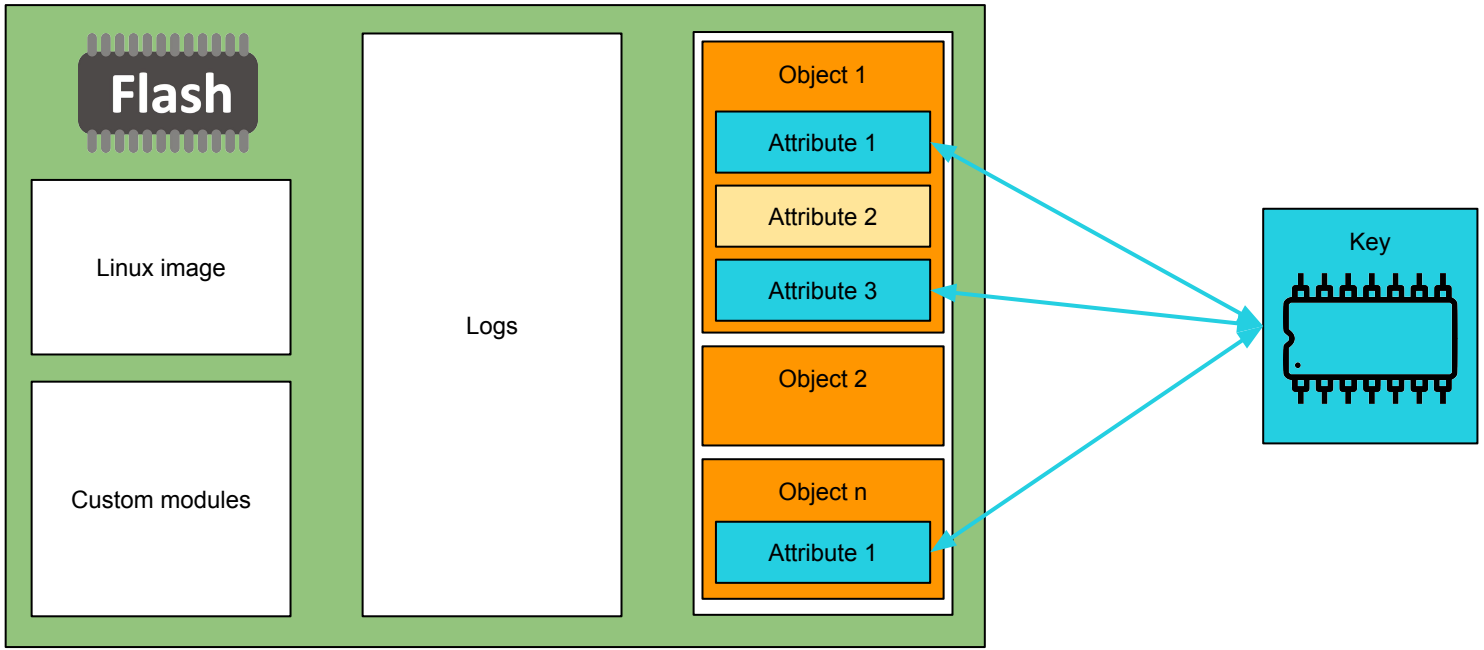
Ledger

- Possibilité d'analyse dynamique
  - Tous les processus tournent avec les privilèges *root*
  - Pas d'options de durcissement activées ni de contre-mesures sécuritaires
- L'intégralité de la flash est accessible via `/dev/mtd`
- Le bootloader est une version légèrement modifiée d'U-Boot :
  - Pas de mécanisme de secure boot

Stockage

- Les données persistantes sont stockées sur une mémoire flash de 64 Mo sur la carte PCI :
  - Image Linux
  - Modules personnalisés
  - Fichiers de journalisation
  - Objets PKCS #11
- Partition dédiée pour les objets PKCS #11 et les informations d’authentification
- Système de fichiers propriétaire
- Les attributs secrets sont stockés chiffrés et déchiffrés à la volée

Stockage


Flash
Linux image
Custom modules
Logs
Object 1
Attribute 1
Attribute 2
Attribute 3
Object 2
Object n
Attribute 1
Key

Première exécution de code

objdump -D API.so | grep memcpy

- ~700 appels à `memcpy` dans `API.so`
- Analyse manuelle pour rechercher les appels avec une taille variable
- `MilenageDerive` est la seule fonction vulnérable
- Mécanisme `CKM_MILENAGE_DERIVE` :
  - Dérivation de clé pour les fonctions MILENAGE *f3*, *f4*, *f5* and *f5**
  - Algorithme d'authentification UMTS

Bug

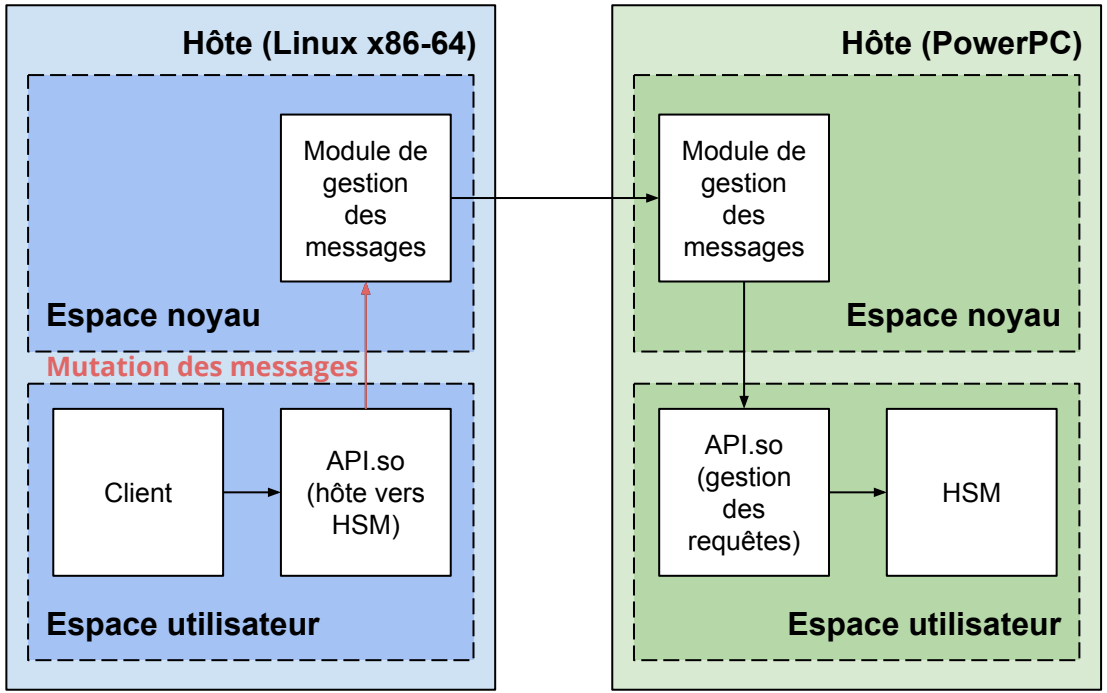
- `CKM_MILENAGE_DERIVE` reçoit un handle sur une MILENAGE de 16 octets, enregistré comme une clé secrète générique
- `MilenageDerive` ne vérifie pas la longueur de la clé secrète :

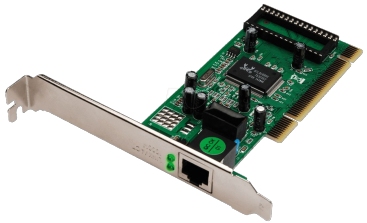
```
int MilenageDerive(...) {
    uint8_t aesKey[16];
    ...
    GetObjectClassAndKeyType(keyObject, &attributeClass, &keyType);
    if (attributeClass == CKO_SECRET_KEY) {
        keyValue = FindAttr(CKA_VALUE, keyObject);
        if (keyValue) {
            valueLen = keyValue->valueLen;
            memcpy(aesKey, keyValue->pValue, keyValue->valueLen);
            ...
}
```

Exploit

- Pas de stack cookie, exécution de code triviale
- Inconvénients :
  - Nécessite d’être authentifié
  - MILENAGE est présent uniquement sur les firmwares récents
  - La stabilité du HSM après l’exécution de code n’est pas garantie

Fuzzing


Hôte (Linux x86-64)
Module de gestion des messages
Espace noyau
Mutation des messages
Client
API.so (hôte vers HSM)
Espace utilisateur
Hôte (PowerPC)
Module de gestion des messages
Espace noyau
API.so (gestion des requêtes)
HSM
Espace utilisateur



Fuzzing

Ledger

- Cible l'implémentation des fonctions PKCS #11
- Mutation aléatoire d'octets
- Difficultés principales :
  - Le module noyau de l'hôte est instable
  - Dénis de service dûs à des OOM sur le HSM
- Résultats :
  - 14 vulnérabilités, plusieurs classes de bugs de corruption mémoire
  - Vulnérabilité similaire à *Heartbleed*
  - Stack et heap overflows
  - etc.

Exemple de vulnérabilité : confusion de type

Découverte du bug

Ledger

```
CK_MECHANISM digestMechanism = { CKM_RIPEMD128, NULL_PTR, 0 };
unsigned char state[4096], data[32];
CK_ULONG ulStateLen;

C_DigestInit(hSession, &digestMechanism);
C_GetOperationState(hSession, state, &ulStateLen);
mutate(state, ulStateLen);
C_SetOperationState(hSession, state, ulStateLen, 0, 0);
C_DigestUpdate(hSession, data, sizeof(data));
```

Analyse du crash

Ledger

- Stacktrace inhabituelle, analyse statique et dynamique
- Bug de confusion de type : une méthode est appelée sur un objet inapproprié
- Conséquences :
  - Fuite mémoire
  - Primitive d’écriture relative
- Pas de contre-mesures ni de durcissement
- Exploit stable

Impact

- Le *payload* est exécuté avec les privilèges root
- Récupération de clés, de la mémoire, de la flash, etc.
- Payload final :
  a. Patch de la fonction de vérification du PIN
  b. Login avec le compte administrateur
  c. Installation d'un module malveillant
  d. Récupération de la flash chiffrée et de la clé de déchiffrement
  e. Déchiffrement *offline*
- L'exploit est un binaire exécuté depuis l'hôte

Contournement de la signature du firmware

Signature du firmware

- Le firmware et les modules sont signés
- Seul le vendeur peut signer le firmware
  - Assure l'intégrité
  - Empêche l'installation d'un firmware personnalisé

- Le mécanisme de mise à jour est implémenté avec des fonctions PKCS #11
- Vulnérabilité : la vérification de signature peut être contournée

Conclusion

Impact

1. D’utilisateur non authentifié à administrateur
    - La vulnérabilité de confusion de type ne nécessite aucun privilèges
    - Exploit stable
2. D’administrateur à une compromission persistante :
    - Installation d’un firmware malveillant avec le contournement de signature
    - L’intégrité du HSM ne peut plus être garantie :
        - Pas de secure boot
        - Downgrade attacks possibles
        - Impossible d’empêcher cette vulnérabilité d’être exploitée à nouveau

Responsible disclosure

- Toutes les vulnérabilités ont été rapportées au constructeur
- Nouveau firmware
- Bulletin de sécurité à diffusion restreinte
- Bindiff pour s’assurer que les correctifs sont corrects

Conclusion

- Résultats… inattendus
- Les certifications concernent principalement les attaques matérielles
- Réduction de la surface d’attaque grâce aux modules personnalisés
- Est-ce que nos attaques fonctionnent sur les versions réseau du HSM ?

Questions ?