Support architectural pour identification de programmes chiffrés dans une architecture sécurisée sans système d'exploitation de confiance

SympA'2008, Fribourg, Suisse

Guillaume DUC guillaume.duc@telecom-bretagne.eu
Ronan KERYELL rk@enstb.org

Institut TÉLÉCOM, TÉLÉCOM Bretagne, Département Informatique,
Lab-STICC/CAC/HPCAS
Plouzané, France

11 février 2008

∃ Besoins de sécurité (I)

Besoins élémentaires :

- Société de l'information ⇝ besoin de confiance en technologies numériques
  - Point de vue utilisateur : pas (peu) de fuites de données personnelles
  - Point de vue architecte système : empêcher attaques
- Applications d'authentification ou de chiffrement style carte à puce
- Routeurs sécurisés
- Programmes et systèmes d'exploitation sécurisés ou secrets

∃ Besoins de sécurité *(II)*

- Installation automatique sécurisée d'ordinateurs en réseau (DHCP, IPsec...)
- Exécution de code réservée à un processeur
- Jeux P2P sans triche
- Protection de contenu multimédia contre le piratage
- Données médicales sensibles qui voyagent partout
- Vote électronique
- Code mobile chiffré et sécurisé (crypto-mobilette)
- SoC complexes avec assemblage de nombreux composants et logiciels avec sécurité non prouvable

Impossible à faire avec des ordinateurs/processeurs classiques ☹

Le plan

Calcul distribué sécurisé ? (I)

- Grilles de calcul (*Grid computing*) sécurisées en milieu naturel (≡ hostile )
  - Point de vue ordinateur client
    -  On ne fait pas confiance à ordinateur distant qui fait tourner son programme... ☹
    - Qu'est-ce qui prouve que l'ordinateur distant est sûr ?
    - Administrateur ou pirate distant peut espionner calculs & données
    - Administrateur ou pirate distant peut modifier les calculs

Exemple pour Seti@Home ou Folding@Home

Répondre « solution non trouvée » au lieu de « solution trouvée »

Calcul distribué sécurisé ? *(II)*

- Point de vue ordinateur serveur
  - Qu'est-ce qui prouve que le programme respecte le contrat ?
  - Qu'est-ce qui prouve que le programme s'exécutant correspond bien à un source donné (logiciel libre...) ?

⇝ Calcul distribué : asymétrie de la confiance...

... souvent oubliée jusqu'à présent 

Construction & compilation application

Hypothèse

∃ Processeur sécurisé spécialisé avec chiffrement code et données

- Compilation classique d'un source
- Édition de lien et chiffrement avec clés secrètes
- Exécutable éventuellement massivement disséminé
- Rajout descripteur contexte d'exécution avec clés secrètes, chiffré avec clé publique processeur cible
  $\rightsquigarrow \approx$ chargeur
- Exécution possible seulement sur le processeur avec bonne clé privée

DRM libres ? (I)

- Pression des éditeurs de contenu pour déployer système de protection des contenus, DRM (*Digital Rights Management*)
- Sujet chaud assez polémique...
  - Pas possible légalement de regarder DVD ou autres avec logiciels libres ☹
  - Par mesure de rétorsion, licence GPL v3 exclut DRM
- Problème intrinsèque : faire tourner programme qui gère des flux secrets
  - Programme qui tourne sur ordinateur cible
    - Rétroingéniérie, debug... possible : accès aux contenus en clair

DRM libres ? *(II)*

- ⇝ Obscurcissement des programmes au niveau source ou binaire pour cacher contenu et fonctionnement
- Pas à l'abri de rétroingéniérie même si pénible
- Assez incompatible avec notion de logiciel libre car accès aux sources ⇝ accès aux secrets...

- Si support matériel pour chiffrer processus, possible de cacher des secrets
- Création d'un processus DRM spécifique à *un* processeur
- Qu'est-ce qui prouve que ce programme secret est bien un DRM et non autre chose ? 

DRM libres ? *(III)*

- ⇝ Besoin moyen sûr d'identifier un programme secret, voire de prouver qu'il est équivalent à un source si logiciel libre
- Pas possible de cacher un secret dans source ⇝ rajout support matériel pour générer secret et clés asymétriques propres au processeur

Le plan

Le plan

Quelques définitions (I)

Definition

Un **processus sécurisé** est un processus dont l'exécution est protégée contre toute action physique ou logique extérieure au processeur et toute action logique interne au processeur. Les espaces mémoire extérieurs au processeur (données, instructions) sont chiffrés et protégés contre les modifications et le bus d'adresse est partiellement randomisé pour éviter des fuites d'information confidentielle et les intrusions logiques

Quelques définitions *(II)*

Definition

L'**exécution sécurisée** d'un processus sécurisé ne peut être que correcte (le processus n'a pas subi d'attaque par modification de son état jusqu'à présent) ou avortée (une attaque active a été détectée et tout état interne concernant le processus est détruit)

Quelques définitions (III)

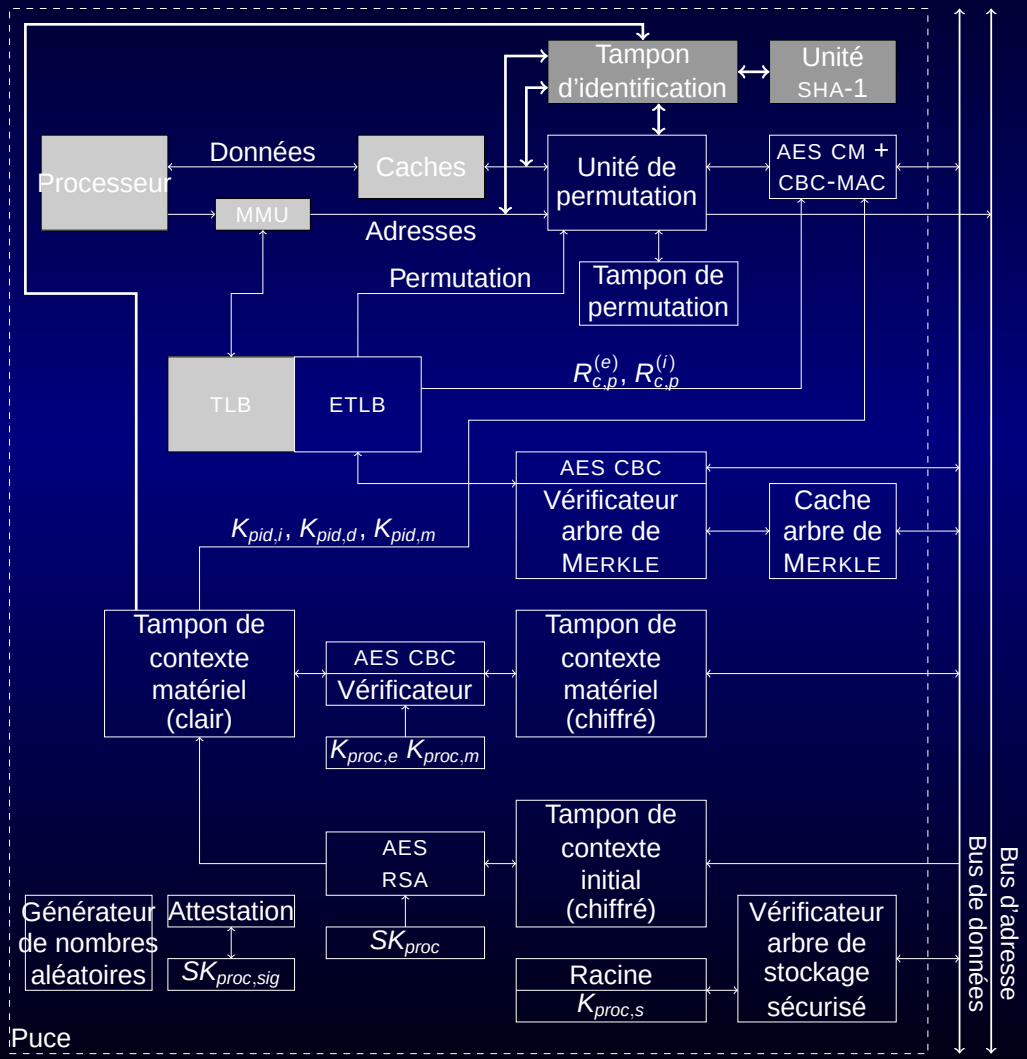
Definition

Un **attaquant** est

- Soit un programme concurrent (processus sécurisé ou pas, système d'exploitation avec tous les privilèges) pouvant espionner ou exécuter des instructions quelconques pour manipuler les états internes du processeur (registres, caches. . .) ou l'extérieur (mémoire, périphériques. . .)
- Soit un humain utilisant des moyens physiques ou logiques pour corrompre ou espionner tout ce qui est extérieur au processeur

Un attaquant peut être passif (espionnage) ou actif (manipulateur)

Architecture CRYPTOPAGE



Le plan

∃ Inconvénient(s) architectures de confiance

Si on a bien travaillé...

- Exécution opaque ☺
- Pas possible d'influencer exécution, à part arrêter exécution ☺
- Pas possible de savoir ce qui tourne 

Besoin

Identifier processus qui tourne à défaut de pouvoir regarder son contenu ☹

Mécanisme d'identification de base

Indentify(p)

$ident_{address} = \perp$

$ident = H(\text{ContexteDéchiffréÉpuré}(p))$

pour $page \in \text{Pages}(p)$

pour $(donnée, adresse) \in \text{Données}(page)$

$ident = H(ident \| donnée \| adresse)$

$ident_{address} = H(ident_{address}, adresse)$

retourne `sign_result`$(ident \| (ident_{address} \stackrel{?}{=} ident_{addressInit}(p)))$

- Le calcul sur $ident_{address}$ permet de vérifier le travail

Délégation à OS non sûr

- Le système d'exploitation ne doit pas pouvoir lire contenu du processus
- Rajouter des instructions dans processeur pour
  - Initialiser calcul empreinte
  - Charger une page ou une ligne depuis mémoire vers processeur, déchiffrer et vérifier intégrité
  - Mettre à jour résumé cryptographique avec adresse et contenu ligne de page préalablement chargée
  - Instruction pour terminer calcul du résumé, attester et renvoyer

  Accèdent aux données indépendamment du mélange des lignes

Algorithme de vérification

Indentify(n)

`spid_init`(n) $\{\ ident_{address} = \bot\ \}$

`spid_hash_ctx` $\{\ ident = H(\text{ContexteDéchiffréÉpuré}(p(n)))\ \}$

pour $page \in \text{Pages}(p)$

`spid_init_page`($page$)

$\{$ *Remplit le tampon d'identification de la page* $\}$

pour $ligne \in \text{Lignes}(page)$

`spid_load_line`($ligne$)

pour $ligne \in \text{Lignes}(page)$

$\{\ ident = H(ident \| donnée \| adresse)\ \}$

$\{\ ident_{address} = H(ident_{address}, adresse)\ \}$

`spid_hash_line`

`spid_hash_page`

retourne `sign_result`($ident \|$

$(ident_{address} \stackrel{?}{=} ident_{addressInit}(p(n))))$

Le plan

Vérification d'un binaire

Comparaison avec un source

- Récupération des sources
- Compilation avec un environnement identique
  - Même compilateur, même bibliothèques, même date et heure, même...
  - Difficile...
  - $\rightsquigarrow$ Machine virtuelle avec environnement préconfiguré... qui pourrait elle-même être identifiée correcte
- Calcul du hachage sur le programme compilé *non* chiffré
- Comparaison avec identification calculée par OS+processeur

Performances

- Contrairement à exécution avec chargement paresseux de pages d'exécutables, identification nécessite lecture de toutes les pages
- Soit un programme de 1 Mo
  - Temps pour charger programme depuis disque 100 Mo/s : 10 ms
  - Temps pour charger programme depuis mémoire 10 Go/s : 100 μs
  - Déchiffrement aligné sur la mémoire : 100 μs
  - Hachage SHA 1 16 Go/s en 45 nm : 62,5 μs
- Même sans recouvrement : 10.2625 ms
- Contraint par vitesse du disque

Optimisation & mémorisation

- Identification associée à un programme
- $\rightsquigarrow$ Identification lors installation programme sur disque
- Cache logiciel associant identifiant du programme à hachage de processus

Travaux en cours (I)

- Projet ANR ARA SSIA SAFESCALE pour concevoir infrastructure de calcul distribué de confiance
  - Amélioration processeur CRYPTOPAGE rajout authentification processus
  - Exécutif d'exécution avec vérification d'exécution probabiliste avec ± CRYPTOPAGE
  - Parallélisation de programmes vers système exécutif KAAPI du LIG (Grenoble) en utilisant PIPS (CDD de 1 an qui démarre)
- Appel d'offre Ministère de la défense pour plateforme maîtrisée SCARABÉE avec Bull, Sagem, LIP6

∃ Projets en rapport

Sujet qui devient à la mode...

- XOM (Berkeley) exécution sécurisée
- AEGIS (MIT) exécution sécurisée et protection contre rejeu
- HIDE (GeorgiaTech) mélange des lignes dans page
- Systèmes parallèles à base de JavaCard (LaBRI)
- Nombreux projets pour l'embarqué (LIRMM, TÉLÉCOM ParisTech Sophia, ex-LESTER)

Pas de moyen d'identification pour comparer à un programme

Conclusion

- De plus en plus difficile de gagner en performance en rajoutant des transistors
  ⇝ En profiter pour en mettre pour la sécurité ! ☺
- Système permettant d'identifier des processus sécurisés
- Retour de la confiance dans le calcul distribué
- Pas besoin d'avoir système d'exploitation de confiance
- Réconciliation DRM et logiciels libres
- ¿ Mercredi après-midi : réunion sur sécurité & grille, SAFESCALE... ?

Table des matières