OSLO : Open Secured LOader

Trusted Computing via l'utilisation d'une puce TPM

Nicolas Stouls

Nicolas.Stouls@lri.fr

Équipe Démons - LRI / Projet ProVal - INRIA-Futurs

Novembre 2007

Plan

OSLO est une pièce logicielle importante pour l'implantation d'une *plateforme de confiance*

1. Qu'est ce qu'une plateforme de confiance
   - Objectifs (sécurité, fonctionnalité)
   - Basé sur l'utilisation d'un chipset *(TPM – Trusted Platform Module)*
   - Implantation dans le cas général
2. Description d'OSLO
   - Présentation générale d'OSLO
   - Description du code
   - Travail réalisé
   - Quelques problèmes rencontrés (et idées de solutions ?)

Plate-forme de confiance

- Idée générale :
  Protéger le logiciel du logiciel
- Objectifs de l'informatique de confiance :
  - Gestion matérielle de la cryptographie
  - Stockage sécurisé
  - Identification distante des logiciels exécutés
    (versonnage, DRM, vérification de licence, etc.)
- Nécessite :
  - Chaîne de confiance
  - Mémorisation des programmes lancés
    (SML – Stored Measurement Log)

Chaîne de confiance

- Objectifs de l'informatique de confiance :
  - **Remote attestation :** Identifier quel logiciel est exécuté à un instant T sur une machine distante
  - **Sealed Memory :** Garantir qu'un secret n'est accessible que par un logiciel particulier
- Principe : La chaîne de confiance

$$A \rightsquigarrow B \rightsquigarrow C$$

  - Un composant est identifié par son hash SHA1 *(Measurement)*
  - Ne jamais rompre la chaîne de confiance
  - Chaque maillon hashe le suivant, l'enregistre puis le lance
    Enregistrement auprès du TPM :
    - *Extension* du TPM
    - Vérification de la confiance
      (SML – Stored Measurement Log)
  - Mais qui mesure le A ?

Chaîne de confiance

- Objectifs de l'informatique de confiance :
  - **Remote attestation :** Identifier quel logiciel est exécuté à un instant T sur une machine distante
  - **Sealed Memory :** Garantir qu'un secret n'est accessible que par un logiciel particulier
- Principe : La chaîne de confiance

$$A \rightsquigarrow B \rightsquigarrow C$$

  - Un composant est identifié par son hash SHA1 *(Measurement)*
  - Ne jamais rompre la chaîne de confiance
  - Chaque maillon hashe le suivant, l'enregistre puis le lance *Enregistrement auprès du TPM :*
    - *Extension* du TPM
    - Vérification de la confiance *(SML – Stored Measurement Log)*
- Mais qui mesure le A ?

Chaîne de confiance

- Objectifs de l'informatique de confiance :
  - **Remote attestation :** Identifier quel logiciel est exécuté à un instant T sur une machine distante
  - **Sealed Memory :** Garantir qu'un secret n'est accessible que par un logiciel particulier
- Principe : La chaîne de confiance

$$A \rightsquigarrow B \rightsquigarrow C$$

  - Un composant est identifié par son hash SHA1 *(Measurement)*
  - Ne jamais rompre la chaîne de confiance
  - Chaque maillon hashe le suivant, l'enregistre puis le lance
    Enregistrement auprès du TPM :
    - *Extension* du TPM
    - Vérification de la confiance
      (SML – Stored Measurement Log)
  - Mais qui mesure le A ?

Racine de confiance statique

- SRTM *(Static Root of Trust for Measurement)* :
  - doit être non modifiable et le premier programme lancé (de préférence intégré au BIOS)
  - Action :
    1. Initialise le TPM
    2. calcule la signature du BIOS
    3. lance le BIOS

SRTM $\rightsquigarrow$ BIOS $\rightsquigarrow$ | Autres firmwares (de cartes d'extension)
Boot loader $\rightsquigarrow$ OS $\rightsquigarrow$ Applications

- Nécessite que le boot loader ne brise pas la chaîne *(Si supérieur à taille MBR alors il doit se hasher)*
- Si doute sur intégrité / correction du BIOS + bootloader : Utiliser d'une racine de confiance dynamique (DRTM)

Racine de confiance statique

- SRTM *(Static Root of Trust for Measurement)* :
  - doit être non modifiable et le premier programme lancé (de préférence intégré au BIOS)
  - Action :
    1. Initialise le TPM
    2. calcule la signature du BIOS
    3. lance le BIOS

SRTM $\rightsquigarrow$ BIOS $\rightsquigarrow$ | Autres firmwares (de cartes d'extension)
Boot loader $\rightsquigarrow$ OS $\rightsquigarrow$ Applications

- Nécessite que le boot loader ne brise pas la chaîne *(Si supérieur à taille MBR alors il doit se hasher)*
- Si doute sur intégrité / correction du BIOS + bootloader : Utiliser d'une racine de confiance dynamique (DRTM)

Racine de confiance dynamique

- Basée sur une *nouvelle* technologie :
  Instructions skinit (AMD) / senter (INTEL)

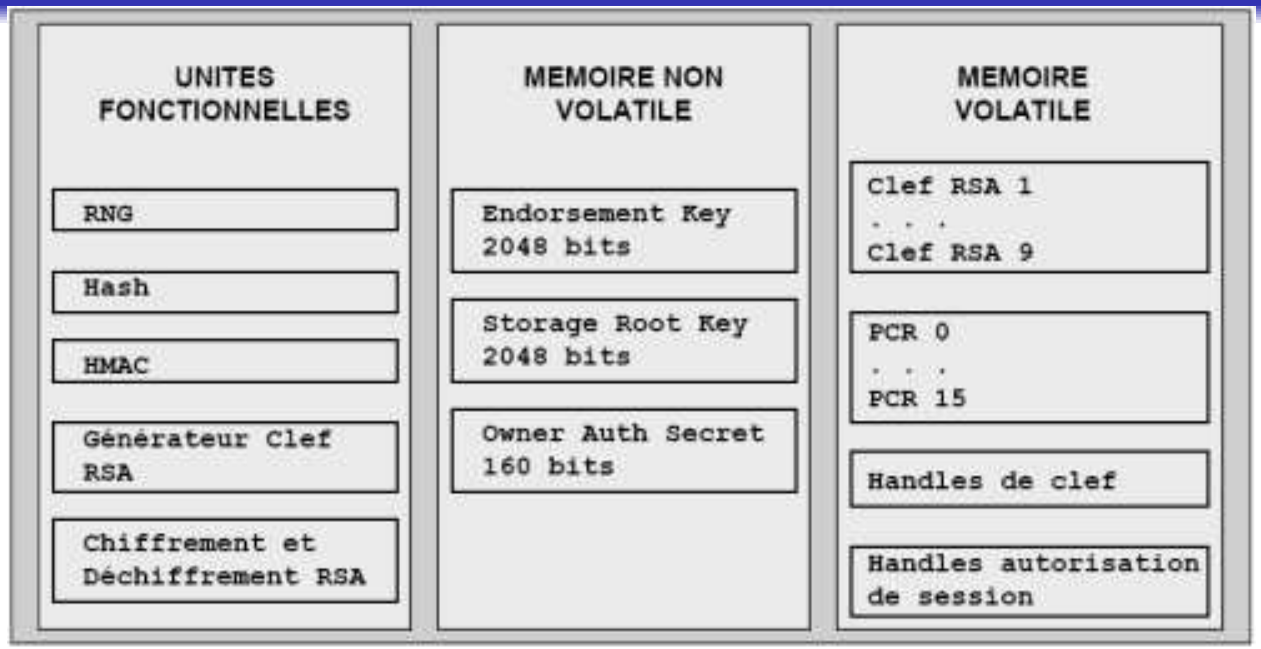
$$\text{DRTM} \rightsquigarrow \text{OS} \rightsquigarrow \text{Applications}$$

- Ne nécessite pas d'être le premier programme lancé
- Lance un Secure Loader *(SL)*

Racine de confiance dynamique

- skinit :
  - réinitialise le processeur
  - envoie un signal infalsifiable de reset au TPM
  - protège la zone mémoire du SL des périphériques
    (l'adresse du SLB est donnée en paramètre dans EAX)
  - synchronise les différents processeurs
  - envoie la signature du SL
  - lance le SL
- SL (Secure Loader) :
  - Contenu dans un SLB (Code + tas) de 64Ko
    si besoins mémoire $\Rightarrow$ protège la zone mémoire voulue
  - Initialise la virtualisation sur le processeur
  - lance un logiciel de confiance *après en avoir vérifié l'identité*

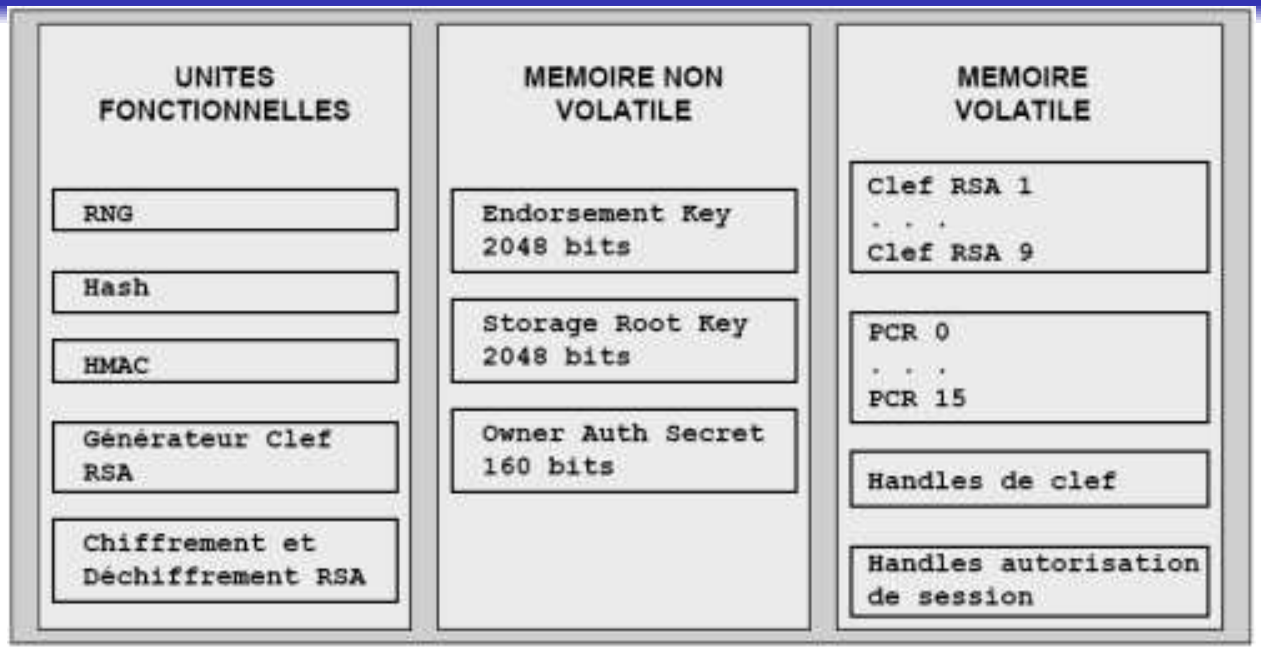
Introduction aux TPM



Gestion matérielle de services cryptographiques :

- Générateur de nombres aléatoires
- Unité de hachage SHA-1
- Gestion et génération de clefs RSA
- Authentification, signature, chiffrement et déchiffrement RSA.

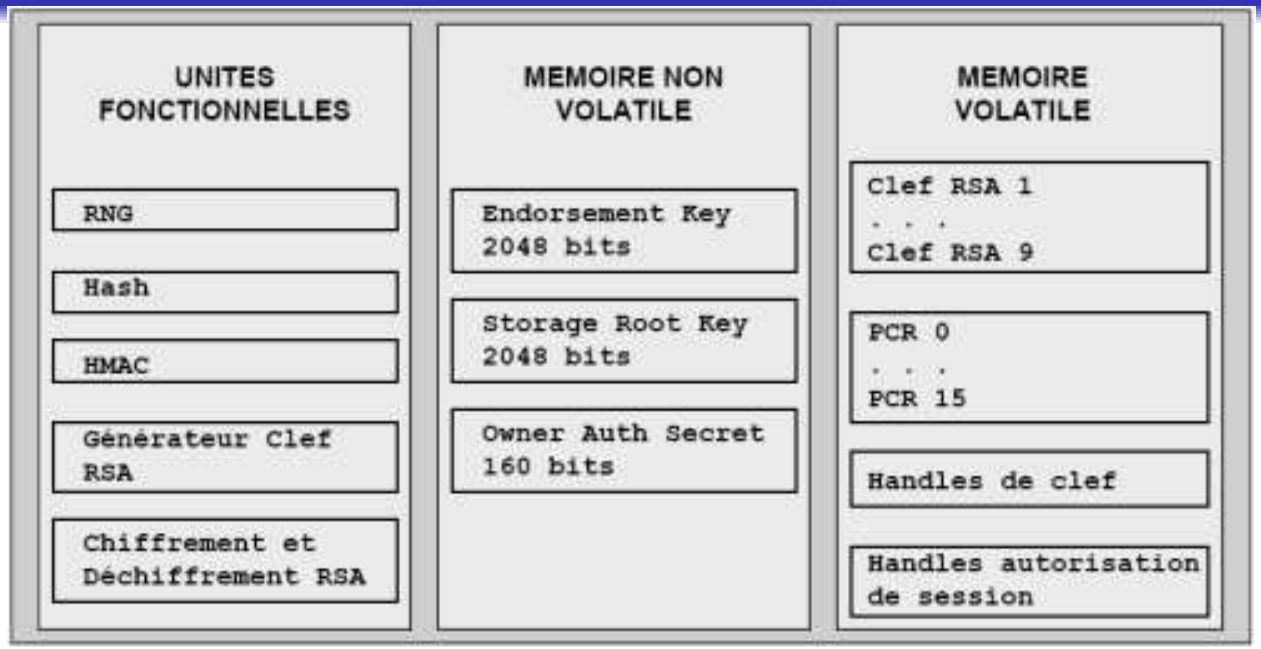
Introduction aux TPM



Gestion de la chaîne de confiance (Registres PCR)

- Registres de 160 bits
- haché d'informations sur la configuration.
  (BIOS, MBR, bootloader, etc.)
- Registres non accessibles en écriture directe (que par extension)
  $Extends(X,PCR_y) \triangleq PCR_y := SHA1(concat(X,PCR_y))$

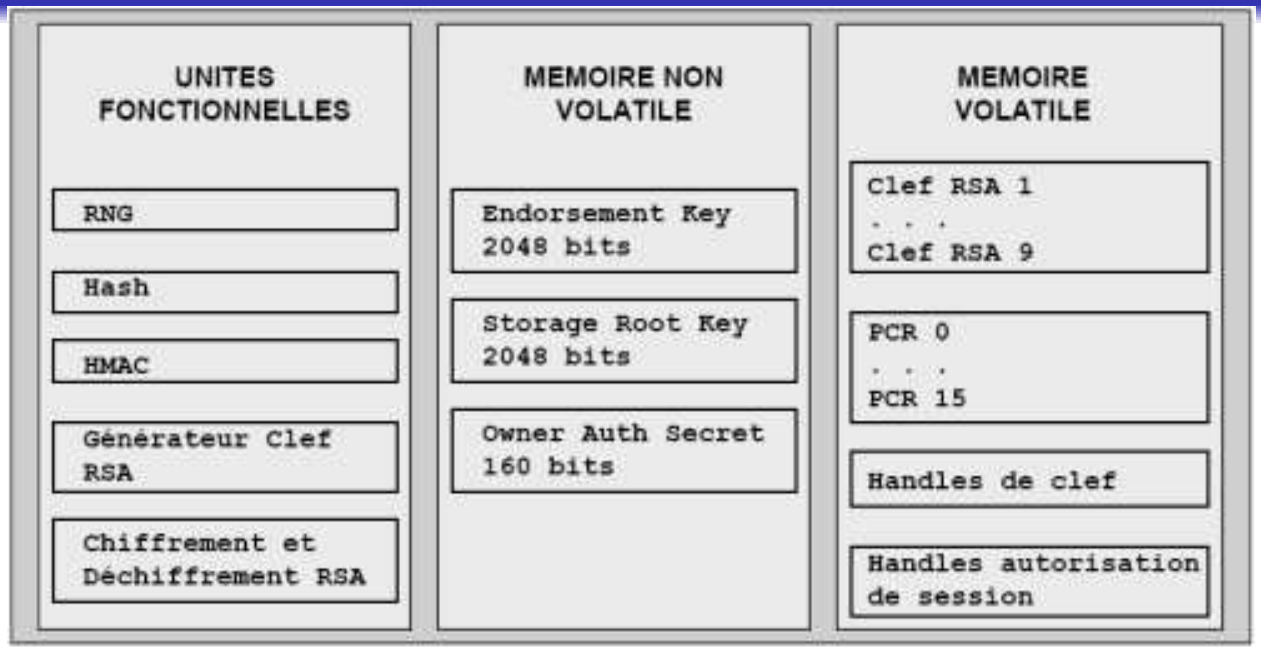
Introduction aux TPM



Premier aparté :

- **Hypothèse de sécurité** :
  impossible de forgé A tq *PCRdesiré = Entends (A)*
- Complexité théorique : 2^{80} opérations de hash
- État de l'art : 2^{69} opérations de hash
  [1] X. Wang, Y.L. Yin and H. Yu, *Finding collisions in the full SHA-1*, Advances in Cryptology–CRYPTO, Springer, 2005

Introduction aux TPM



TPM = composant passif (ni contrôle, ni monitoring)

- Mesures réalisées par le PC et envoyées au TPM
- Il ne sais pas quelles données sont mesurées
- Il ne peux pas réinitialiser l'ordinateur

Interactions avec un TPM

- Dépend de la plate-forme (PC, palm, téléphone, etc.).
- Sur x86 :
  - Virtualisation du processeur obligatoire
  - Accès par mémoire partagée (MMIO)
  - Adresse fixe 0xFED40000
  - 5 pages mémoires de 4Ko protégées :

0xFED40xxx	0x0xxx
0xFED41xxx	0x1xxx
0xFED42xxx	0x2xxx
0xFED43xxx	0x3xxx
0xFED44xxx	0x4xxx

Présentation générale d'OSLO

- OSLO est un DRTM *(Dynamic Root of Trust for Measurement)*
  Origine d'une chaîne de confiance avec racine dynamique
  - Lancé par un boot loader *(par exemple GRUB)*
  - Initialise le TPM
  - Contient un Secure Loader *(SL)* lancé par skinit
  - Contient 3 programmes pouvant être lancés par le SL :
    - ***Beirut :*** *hashe une ligne de commande et l'exécute*
    - ***Pamplona :*** *Annule le lancement du TPM*
    - ***Munich :*** *Lance un Linux*

- Logiciel très proche du matériel
  - Développement très compact
  - Routines assembleur
  - Adresses mémoire accédées écrites en "dur"
    (MMIO – Memory Mapped I/O)
  - Code inliné
  - Conversions explicites little-endian/big-endian

Présentation générale d'OSLO

- OSLO est un DRTM *(Dynamic Root of Trust for Measurement)*
  Origine d'une chaîne de confiance avec racine dynamique
  - Lancé par un boot loader *(par exemple GRUB)*
  - Initialise le TPM
  - Contient un Secure Loader *(SL)* lancé par skinit
  - Contient 3 programmes pouvant être lancés par le SL :
    - ***Beirut :*** *hashe une ligne de commande et l'exécute*
    - ***Pamplona :*** *Annule le lancement du TPM*
    - ***Munich :*** *Lance un Linux*

- Logiciel très proche du matériel
  - Développement très compact
  - Routines assembleur
  - Adresses mémoire accédées écrites en "dur"
    (MMIO – Memory Mapped I/O)
  - Code inliné
  - Conversions explicites little-endian/big-endian

Description du DRTM OSLO

- osl : DRTM (lanceur de skinit + corps du Secure Loader).
- asm : Routines assembleur
- elf : Chargeur d'exécutable
- mp : start/stop multi-processeurs
- sha : Primitives de hashage
- tis : Primitives d'accès TPM
- tpm : API haut niveau d'accès TPM
- util : I/O (text/port com) + gestion assertions

Description du DRTM OSLO

- osl : DRTM (lanceur de skinit + corps du Secure Loader).
- asm : Routines assembleur
- elf : Chargeur d'exécutable
- mp : start/stop multi-processeurs
- sha : Primitives de hashage
- tis : Primitives d'accès TPM
- tpm : API haut niveau d'accès TPM
- util : I/O (text/port com) + gestion assertions

Spécifications décrites / problèmes rencontrés

- asm :
  - Spécification basique (Pré=true/assigns=\nothing)
  - Post-conditions pas toujours descriptibles *(in/out)*
  - Post-conditions n'ont pas toujours de sens *(reboot/jmp/skinit)* `__attribute__((noreturn))`

Spécifications décrites

- util :
  - Spécification basique (Pré=true/assigns=\nothing)
  - Axiomatisation des chaînes de caractères
  - Exploitation des macro de debug `Assert` et `Error`
    permet de prouver l'absence de violation d'erreur
  - Accès à la mémoire MMIO
  - Communication avec les ports I/O

Exemple de spécification simple

Exemple d'affichage d'un caractère

```
void out_char (unsigned value){
  static unsigned int col=0 ;
  if (value !='\n') {
    unsigned short *p = ((unsigned short *) (0xb8000+24*160))+col ;
    *p = 0x0f00 | value ;
    col++ ;
  }
  if (col>=80 || value == '\n') {
    unsigned short *p = ((unsigned short *) 0xb8000) ;
    col=0 ;
    memcpy(p, p+80, 24*160) ;
    memset(((unsigned short *) (0xb8000+24*160)), 0, 160) ;
  }
}
```

Exemple de spécification simple

Exemple d'affichage d'un caractère

```
void out_char (unsigned value){
  static unsigned int col=0 ;  /* Pb caduceus #4352 */
  if (value !='\n') {
    unsigned short *p = ((unsigned short *) (0xb8000+24*160))+col ;
    *p = 0x0f00 | value ;
    col++ ;
  }
  if (col>=80 || value == '\n') {
    unsigned short *p = ((unsigned short *) 0xb8000) ;
    col=0 ;
    memcpy(p, p+80, 24*160) ;
    memset(((unsigned short *) (0xb8000+24*160)), 0, 160) ;
  }
}
```

Exemple de spécification simple

Exemple d'affichage d'un caractère

```
void out_char (unsigned value){
  static unsigned int col=0 ;  /* Pb caduceus #4352 */
  if (value !='\n') {
    unsigned short *p = ((unsigned short *) (0xb8000+24*160))+col ;
 /* Pas d'@ en dur */
    *p = 0x0f00 | value ;
    col++ ;
  }
  if (col>=80 || value == '\n') {
    unsigned short *p = ((unsigned short *) 0xb8000) ;
    col=0 ;
    memcpy(p, p+80, 24*160) ;
    memset(((unsigned short *) (0xb8000+24*160)), 0, 160) ;
  }
}
```

Exemple de spécification simple

Exemple d'affichage d'un caractère

```
void out_char (unsigned value){
  static unsigned int col=0 ;  /* Pb caduceus #4352 */
  if (value !='\n') {
    unsigned short *p = ((unsigned short *) (0xb8000+24*160))+col ;
 /* Pas d'@ en dur */
    *p = 0x0f00 | value ;  /* Validité adresse à établir */
    col++ ;
  }
  if (col>=80 || value == '\n') {
    unsigned short *p = ((unsigned short *) 0xb8000) ;
    col=0 ;
    memcpy(p, p+80, 24*160) ;
    memset(((unsigned short *) (0xb8000+24*160)), 0, 160) ;
  }
}
```

Exemple de spécification simple

Exemple d'affichage d'un caractère

```
void out_char (unsigned value){
  static unsigned int col=0 ;  /* Pb caduceus #4352 */
  if (value !='\n') {
    unsigned short *p = ((unsigned short *) (0xb8000+24*160))+col ;
 /* Pas d'@ en dur */
    *p = 0x0f00 | value ;  /* Validité adresse à établir */
    col++ ;
  }
  if (col>=80 || value == '\n') {
    unsigned short *p = ((unsigned short *) 0xb8000) ;
    col=0 ;
    memcpy(p, p+80, 24*160) ;  /* Violation spécifications publiques */
    memset(((unsigned short *) (0xb8000+24*160)), 0, 160) ;
  }
}
```

Exemple de spécification simple

Exemple d'affichage d'un caractère

```
void out_char (unsigned value){
  static unsigned int col=0 ;  /* Pb caduceus #4352 */
  if (value !='\n') {
    unsigned short *p = ((unsigned short *) (0xb8000+24*160))+col ;
 /* Pas d'@ en dur */
    *p = 0x0f00 | value ;  /* Validité adresse à établir */
    col++ ;
  }
  if (col>=80 || value == '\n') {
    unsigned short *p = ((unsigned short *) 0xb8000) ;
    col=0 ;
    memcpy(p, p+80, 24*160) ;  /* Violation spécifications publiques */
    memset(((unsigned short *) (0xb8000+24*160)), 0, 160) ;  /*
Offset 8bits / type 16bits */
  }
}
```

Exemple de contournement

Exportation adresse MMIO - Par opération

```
/*@ requires \true
  @ assigns \nothing
  @ ensures \valid_range(\result,0,65535)
  */
unsigned short *GetAdrMemVid(){
  return ((unsigned short *) 0xb8000) ;
}
```

Limitation : *fonctions non utilisables dans specs caduceus*

Exportation adresse MMIO - Par variable

```
static unsigned short *AdrMemVid=((unsigned short *) 0xb8000) ;
//@ invariant I1 : \valid_range(AdrMemVid,0,65535)
```

Limitation : *Initialisation syntaxiquement incorrecte*

Exemple de contournement

Exportation adresse MMIO - Par opération

```
/*@ requires \true
  @ assigns \nothing
  @ ensures \valid_range(\result,0,65535)
  */
unsigned short *GetAdrMemVid(){
  return ((unsigned short *) 0xb8000) ;
}
```

Limitation : *fonctions non utilisables dans specs caduceus*

Exportation adresse MMIO - Par variable

```
static unsigned short *AdrMemVid=((unsigned short *) 0xb8000) ;
//@ invariant I1 : \valid_range(AdrMemVid,0,65535)
```

Limitation : *Initialisation syntaxiquement incorrecte*

Problèmes de modularité

- Constante déclarée dans l'entête :

  extern const char *message_label ;
- Constante instanciée dans chaque exécutable
- Comment garantir que is_string(message_label) ?
- Palliatif temporaire :

  *//@ **invariant** I2 : \valid(message_label)*

Problèmes de généricité des fonctions

- Les 4 fonctions suivantes :

```
          char *memcpy_char   (char *dest          , const char *src          , long n);
unsigned char *memcpy_uchar  (unsigned char *dest , const unsigned char *src , long n);
         short *memcpy_short  (short *dest         , const short *src         , long n);
unsigned short *memcpy_ushort (unsigned short *dest , const unsigned short *src , long n);
```

- peuvent être remplacées par :

```
void *memcpy (void *dest , const void *src , long n);
```

- Problème : exprimer et vérifier la post-condition :

```
//@ ensures \result==dest && (\forall int i ; 0<=i<n => *((char* )dest)+i) == *((char *) src)+i)))
```

Préservation de l'encapsulation

Header

```
/*@ requires \true
  @ assigns \nothing
  @ ensures \true
  */
F();
```

Fichier c

```
static int x=0;
//@ invariant I3 : 0<=x<=316
/*@ assigns_local x
  @ ensures_local x<=127
  */
F(){
  if (x<127) x++;
  else x :=0;
}
```

- Renforcement possible de la post
- Utilisation dans la Pré d'opérations locales

FAQ

- L'application en cours est elle celle enregistrée ?
  Il faut une "complicité" de l'OS ou que le SL soit utilisé pour tout chargement de code exécutable (??)
- Toute application peut elle être chargée, même un virus ?
  Oui. Mais cela brise la chaîne et annule les droits d'accès au TPM
- Quelles hypothèses sur communications MMIO ?
  Il semblerait que tout envoie vers le périphérique ne soit pas systématiquement envoyé tout de suite par le processeur.
  `http://iou.parisc-linux.org/ols_2002/4_3MMIO_is_harder.html`

FAQ

- L'application en cours est elle celle enregistrée ?
  Il faut une "complicité" de l'OS ou que le SL soit utilisé pour tout chargement de code exécutable (??)
- Toute application peut elle être chargée, même un virus ?
  Oui. Mais cela brise la chaîne et annule les droits d'accès au TPM
- Quelles hypothèses sur communications MMIO ?
  Il semblerait que tout envoie vers le périphérique ne soit pas systématiquement envoyé tout de suite par le processeur.
  `http://iou.parisc-linux.org/ols_2002/4_3MMIO_is_harder.html`

FAQ

- L'application en cours est elle celle enregistrée ?
  Il faut une "complicité" de l'OS ou que le SL soit utilisé pour tout chargement de code exécutable (??)
- Toute application peut elle être chargée, même un virus ?
  Oui. Mais cela brise la chaîne et annule les droits d'accès au TPM
- Quelles hypothèses sur communications MMIO ?
  Il semblerait que tout envoie vers le périphérique ne soit pas systématiquement envoyé tout de suite par le processeur.

  `http://iou.parisc-linux.org/ols_2002/4_3MMIO_is_harder.html`

Propositions pour les outils

- Message d'erreur (warning ?) si oubli de spéc de boucle (Actuellement, l'absence de spec de boucle génère simplement des OPs infaisables).
- Autoriser le cast de tableaux ou de structures
  Attention à la gestion little-endian big-endian
  Exemple : x86 = little-endian / TPM = big-endian
- Accepter les opérations inlinées
  (mettre pré/post en asserts ?)

Propositions pour les outils

- Message d'erreur (warning ?) si oubli de spéc de boucle (Actuellement, l'absence de spec de boucle génère simplement des OPs infaisables).
- Autoriser le cast de tableaux ou de structures
  Attention à la gestion little-endian big-endian
  Exemple : x86 = little-endian / TPM = big-endian
- Accepter les opérations inlinées
  (mettre pré/post en asserts ?)

Propositions pour les outils

- Message d'erreur (warning ?) si oubli de spéc de boucle (Actuellement, l'absence de spec de boucle génère simplement des OPs infaisables).
- Autoriser le cast de tableaux ou de structures
  Attention à la gestion little-endian big-endian
  Exemple : x86 = little-endian / TPM = big-endian
- Accepter les opérations inlinées
  (mettre pré/post en asserts ?)

Propriétés intéressantes à vérifier

- le DRTM de OSLO charge bien la signature complète du code qui s'exécute après lui
- Munich charge bien la signature complète du code qui s'exécute après lui
- Si un programme de confiance est modifié pendant qu'il est en mémoire, alors il perd la main sur le TPM

Propriétés intéressantes à vérifier

- le DRTM de OSLO charge bien la signature complète du code qui s'exécute après lui
- Munich charge bien la signature complète du code qui s'exécute après lui
- Si un programme de confiance est modifié pendant qu'il est en mémoire, alors il perd la main sur le TPM

Propriétés intéressantes à vérifier

- le DRTM de OSLO charge bien la signature complète du code qui s'exécute après lui
- Munich charge bien la signature complète du code qui s'exécute après lui
- Si un programme de confiance est modifié pendant qu'il est en mémoire, alors il perd la main sur le TPM

Propriétés intéressantes à vérifier

- le DRTM de OSLO charge bien la signature complète du code qui s'exécute après lui
- Munich charge bien la signature complète du code qui s'exécute après lui
- Si un programme de confiance est modifié pendant qu'il est en mémoire, alors il perd la main sur le TPM

Avez vous des questions ?

Merci de votre attention.