Compromission d'une application bancaire JavaCard par attaque logicielle

J. Lancia

SERMA Technologies, CESTI

Symposium sur la sécurité des technologies de l'information et des communications, 2012

Plan

Outline

Présentation du produit attaqué

Plateforme Java Card 2.2.2

- Machine virtuelle à faible empreinte mémoire dédiée aux cartes à puces
- Applications codées avec un sous-ensembe de Java

Applet bancaire embarquée

- Protège les biens sécuritaires : PIN, clés, algorithmes cryptographiques
- Cible de notre expertise

Plateforme ouverte

- Possibilité de charger des applets en phase opérationelle
- Ouvre la voie aux attaques logiques

Présentation du produit attaqué

Plateforme Java Card 2.2.2

- Machine virtuelle à faible empreinte mémoire dédiée aux cartes à puces
- Applications codées avec un sous-ensembe de Java

Applet bancaire embarquée

- Protège les biens sécuritaires : PIN, clés, algorithmes cryptographiques
- Cible de notre expertise

Plateforme ouverte

- Possibilité de charger des applets en phase opérationelle
- Ouvre la voie aux attaques logiques

Présentation du produit attaqué

Plateforme Java Card 2.2.2

- Machine virtuelle à faible empreinte mémoire dédiée aux cartes à puces
- Applications codées avec un sous-ensembe de Java

Applet bancaire embarquée

- Protège les biens sécuritaires : PIN, clés, algorithmes cryptographiques
- Cible de notre expertise

Plateforme ouverte

- Possibilité de charger des applets en phase opérationelle
- Ouvre la voie aux attaques logiques

Règles du jeu

Contraintes

- Temps contraint : 15 jours
- Nombre d'échantillons contraint : 20 échantillons

Atouts

- Clés de chargement d'applets disponibles
- Code disponible (applet et plateforme)...
- ... vraiment nécessaire ?

Règles du jeu

Contraintes

- Temps contraint : 15 jours
- Nombre d'échantillons contraint : 20 échantillons

Atouts

- Clés de chargement d'applets disponibles
- Code disponible (applet et plateforme)...
- ... vraiment nécessaire ?

Outline

Sécurité des plateformes Java Card

Sécurité du code

- Vérification statique de bytecode
- Vérifications dynamiques (ex. cast)

Vérification de bytecode

La vérification de bytecode est faite offcard. Il est possible de modifier le bytecode entre la vérification et le chargement.

Sécurité des plateformes Java Card

Sécurité du code

- Vérification statique de bytecode
- Vérifications dynamiques (ex. cast)

Vérification de bytecode

La vérification de bytecode est faite offcard. Il est possible de modifier le bytecode entre la vérification et le chargement.

Sécurité des plateformes Java Card

Partage d'objets entre applets

- Notion de propriété d'instance
- Tous les partages entre applets sont contôlés par le firewall

Même quand tout se passe mal

- Un attaquant obtient une référence sur une instance d'une autre applet
- Le firewall bloque l'accès

Sécurité des plateformes Java Card

Partage d'objets entre applets

- Notion de propriété d'instance
- Tous les partages entre applets sont contôlés par le firewall

Même quand tout se passe mal

- Un attaquant obtient une référence sur une instance d'une autre applet
- Le firewall bloque l'accès

Sécurité additionnelle du produit

Pile typée

- Représentation interne différente pour les références et les types natifs

Méta-données sur les objets

- Informations supplémentaires gérées par la machine virtuelle
- Permet des vérifications supplémentaires pendant l'exécution
- Nombre de champs, classe de l'objet

Sécurité additionnelle du produit

Pile typée

- Représentation interne différente pour les références et les types natifs

Méta-données sur les objets

- Informations supplémentaires gérées par la machine virtuelle
- Permet des vérifications supplémentaires pendant l'exécution
- Nombre de champs, classe de l'objet

Sécurité additionnelle du produit

Chiffrement par applet

- Les données sensibles (clés, PIN ...) sont stockées chiffrées
- La clé de chiffrement est différente pour chaque applet

Même quand tout se passe mal

- Un attaquant lit toute la mémoire d'une autre applet
- Les données sensibles restent protégées

Que va-t-on bien pouvoir faire ?

Sécurité additionnelle du produit

Chiffrement par applet

- Les données sensibles (clés, PIN ...) sont stockées chiffrées
- La clé de chiffrement est différente pour chaque applet

Même quand tout se passe mal

- Un attaquant lit toute la mémoire d'une autre applet
- Les données sensibles restent protégées

Que va-t-on bien pouvoir faire ?

Outline

Confusion de type

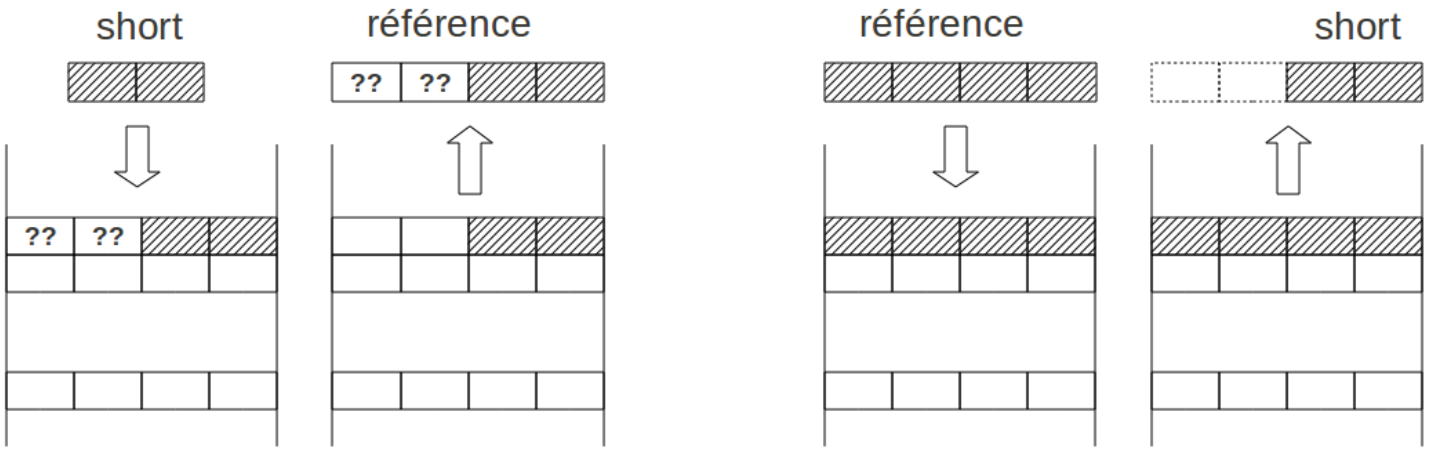
Principe

- Convertion illégale entre objets de types incompatibles

Accéder à un objet comme un short

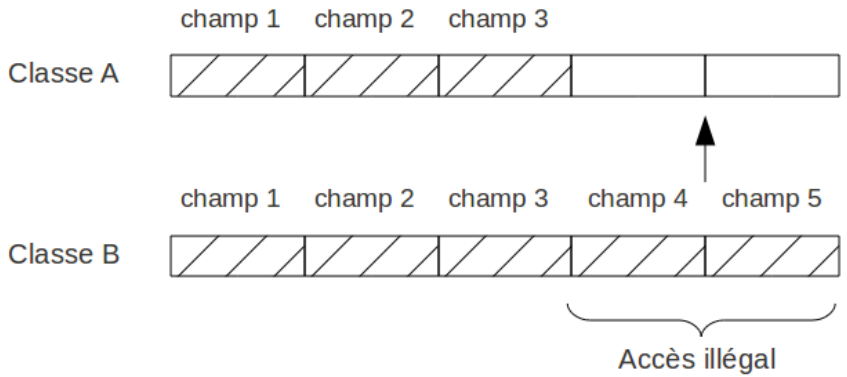
- Forger des références vers les objets d'autres applets
- Lire la représentation interne des objets

- Impossible à cause de la pile typée



Accéder à un objet comme un objet d'un autre type

- Lire au delà des limites de l'objet
- Impossible à cause des meta-données

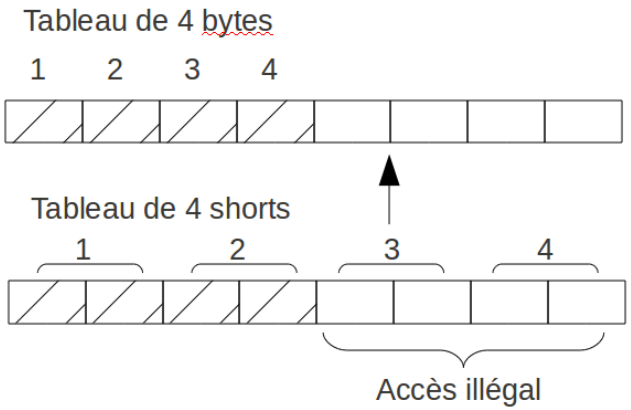


Accéder à un tableau de shorts comme un tableau de bytes

- Les bytes sont sur 1 octet, les shorts sont sur 2 octets
- Lire au delà des limites du tableau

Pas détecté par la VM

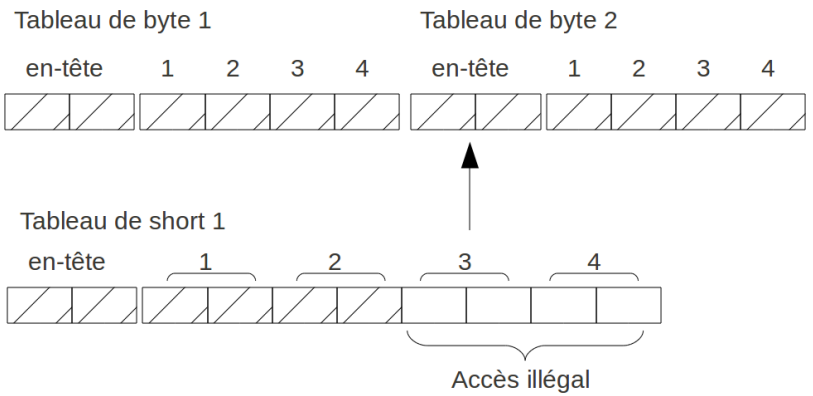
- Tous les tableaux en Java sont de la classe Objet
- Les types natifs ne sont pas distingués dans les méta données



Outline

Allocation indirecte

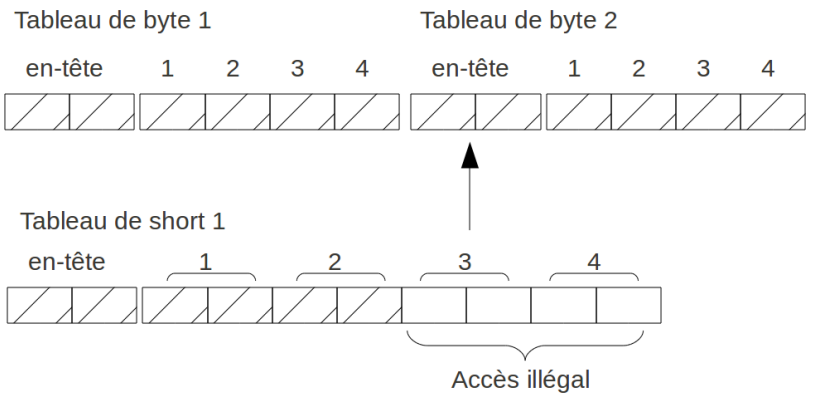
- Allocation de deux tableaux de bytes successifs
- Confusion de type sur le premier tableau
  - Accès illégal à l'en-tête du 2° tableau
- Modification de la taille dans l'en-tête du 2° tableau



Accès illégal à 32 kB maximum en lecture/écriture

Allocation indirecte

- Allocation de deux tableaux de bytes successifs
- Confusion de type sur le premier tableau
  - Accès illégal à l'en-tête du 2° tableau
- Modification de la taille dans l'en-tête du 2° tableau



Accès illégal à 32 kB maximum en lecture/écriture

Accès mémoire illégal

Lecture

- L'AID de l'application bancaire est présent dans les données
- Analyse en rétro-conception des octets lus
- Les données sensibles sont chiffrées par applet
  - Pas accessibles par lecture du tableau

Ecriture

- Modification du code de l'application bancaire

Outline

Présentation de l'attaque

PIN

- Donnée critique de l'application bancaire
- Permet d'authentifier le porteur de la carte

Objectif

- Modifier le code de la carte pour qu'elle accepte un PIN incorrect

Présentation de l'attaque

PIN

- Donnée critique de l'application bancaire
- Permet d'authentifier le porteur de la carte

Objectif

- Modifier le code de la carte pour qu'elle accepte un PIN incorrect

Retro conception du code de VERIFY

- Bytecode

```
L3 : sload 4;
     slookupswitch
     L6 2 90 L5 165 L4;
L4 : getfield_a_this 0;
     invokevirtual 18;
     return;
L5 : sspush 25536;
     getfield_a_this 0;
     invokevirtual 19;
     sor;
     invokestatic 20;
L6 : sspush -8531;
     invokestatic 20;
     return;
```

- Java Card

```
result=OwnerPIN.check(pin,offset,SIZE);
...
switch (result)
{
  case TRUE:
       resetPTC();
       return;
  case FALSE:
       UserException.throwIt
          ((short)(0x63C0|(PTC[0])));
  default:
       break;
}
ISOException.throwIt(Terminate);
```

Modification du code de VERIFY

Traitement du résultat de la vérification de PIN

slookupswitch L6 2 90 L5 165 L4;

Code original

- 90 L5 = Cas défavorable
  - Si la variable vaut 0x5A, jump à L5
- 165 L4 = Cas favorable
  - Si la variable vaut 0xA5, jump à L4

Code modifié

- 90 L4 = Cas **FAVORABLE**
- 165 L4 = Cas favorable

Modification du code de VERIFY

Traitement du résultat de la vérification de PIN

slookupswitch L6 2 90 L5 165 L4;

Code original

- 90 L5 = Cas défavorable
  - Si la variable vaut 0x5A, jump à L5
- 165 L4 = Cas favorable
  - Si la variable vaut 0xA5, jump à L4

Code modifié

- 90 **L4** = Cas **FAVORABLE**
- 165 L4 = Cas favorable

Code de VERIFY après l'attaque

```
result=OwnerPIN.check(pin,offset,SIZE);
switch (result)
{
   case TRUE:
   case FALSE:
        resetPTC();
        return;
   default:
        break;
}
ISOException.throwIt(Terminate);
```

Sortie de la commande VERIFY

- Quel que soit le code PIN présenté :
  - La carte émet une réponse positive
  - Le compteur d'essais est remis à 0

Outline

Présentation de l'attaque

clés Kmac Kenc

- Données critiques de l'application bancaire
- Permet d'authentifier la banque
  - Modification du PIN, mise à zero des compteurs de transaction
  - Validation des transactions

Objectif

- Modifier le code de la carte pour accéder aux clés

Plan d'attaque

- Déterminer les identifiants de champs correspondant aux clés
- Modification d'une portion de code qui renvoie des données pour renvoyer les clés

Présentation de l'attaque

clés Kmac Kenc

- Données critiques de l'application bancaire
- Permet d'authentifier la banque
  - Modification du PIN, mise à zero des compteurs de transaction
  - Validation des transactions

Objectif

- Modifier le code de la carte pour accéder aux clés

Plan d'attaque

- Déterminer les identifiants de champs correspondant aux clés
- Modification d'une portion de code qui renvoie des données pour renvoyer les clés

Présentation de l'attaque

clés Kmac Kenc

- Données critiques de l'application bancaire
- Permet d'authentifier la banque
  - Modification du PIN, mise à zero des compteurs de transaction
  - Validation des transactions

Objectif

- Modifier le code de la carte pour accéder aux clés

Plan d'attaque

- Déterminer les identifiants de champs correspondant aux clés
- Modification d'une portion de code qui renvoie des données pour renvoyer les clés

Où sont mes clés ?

Quand tout semble aller de travers, l'application exécute le code suivant :

```
getfield_a_this 2;
invokeinterface 1 13 0;
getfield_a_this 3;
invokeinterface 1 13 0;
getfield_a_this 4;
invokeinterface 1 13 0;
```

On en déduit qu'il s'agit du code d'effacement des clés :

```
SMI_K.clearKey();
SMC_K.clearKey();
AC_K.clearKey();
```

Les identifiants des clés sont 2, 3, 4

Où sont mes clés ?

Quand tout semble aller de travers, l'application exécute le code suivant :

```
getfield_a_this 2;
invokeinterface 1 13 0;
getfield_a_this 3;
invokeinterface 1 13 0;
getfield_a_this 4;
invokeinterface 1 13 0;
```

On en déduit qu'il s'agit du code d'effacement des clés :

```
SMI_K.clearKey();
SMC_K.clearKey();
AC_K.clearKey();
```

Les identifiants des clés sont 2, 3, 4

Où sont mes clés ?

Quand tout semble aller de travers, l'application exécute le code suivant :

```
getfield_a_this 2;
invokeinterface 1 13 0;
getfield_a_this 3;
invokeinterface 1 13 0;
getfield_a_this 4;
invokeinterface 1 13 0;
```

On en déduit qu'il s'agit du code d'effacement des clés :

```
SMI_K.clearKey();
SMC_K.clearKey();
AC_K.clearKey();
```

Les identifiants des clés sont 2, 3, 4

Extraction des clés

méthode getKey de la classe DESKey

- byte getKey (byte [] keyData , short kOff)

méthode generateData de la classe RandomData

- void generateData (byte [] buffer , short offset , short length)

- Similarités :
  - remplissent un tableau de bytes passés en argument
  - acceptent un offset
- Différence :
  - generateData a un argument de longueur

Retro conception du code de GET CHALLENGE

Génération d'aléa dans la méthode GET CHALLENGE

random.generateData(tmpbuffer,(short)0,size);

Bytecode

```
getfield_a_this 1;
getfield_a_this 7;
sconst_0;
sload_2;
invokevirtual 22;
```

```
charge random
charge tmpbuffer
charge (short)0
charge size
invocation de generateData
```

Modification du code de GET CHALLENGE

Lecture de la clé

- byte getKey (byte [] keyData , short kOff)

Bytecode

```
> getfield_a_this 2;
getfield_a_this 7;
sconst_0;
> nop;
> invokeinterface 4;
```

```
charge SMI_K à la place de random
charge tmpbuffer
charge (short)0
ne charge pas size
invocation de getKey
```

Code de GET CHALLENGE après l'attaque

Code Java Card

```
SMI_K.getKey(tmpbuffer,(short)0);
Util.arrayCopyNonAtomic(tmpbuffer,
        (short)0,apduBuffer,(short)0, size);
apdu.setOutgoingLength(size);
apdu.sendBytes((short)0,size);
```

Sortie de la commande GET CHALLENGE

- L'application bancaire renvoie les 8 premiers octets de la clé de protection en intégrité Kmac
- Les 8 octets suivants peuvent être obtenus en modifiant l'offset

Rappel

Chiffrement par applet

- Les données sensibles (clés, PIN...) sont stockées chiffrées
- La clé de chiffrement est différente pour chaque applet

Même quand tout se passe mal

- Un attaquant lit toute la mémoire d'une autre applet
- Les données sensibles restent protégées

Rappel

Chiffrement par applet

- Les données sensibles (clés, PIN...) sont stockées chiffrées
- La clé de chiffrement est différente pour chaque applet

Et pourtant

- C'est l'applet compromise qui lit ses clés...
- ... et nous les donne !

Réalisation d'un YES Card

Récupération des clés cryptographiques

Outline

Bref j'ai craqué une carte à puces

Un produit bien sécurisé

- Sécurité de Java Card
- Sécurité additionnelle du produit

Attaque par confusion de type

- Accès en lecture/écriture au code de l'applet

Compromission de l'applet bancaire

- PIN
- clés cryptographiques
- et potentiellement toutes les données de l'application

Outline

Réalisme de l'attaque

Code malicieux

- Le code de l'applet d'attaque ne passe pas le bytecode verifier
- Impossible à embarquer dans un applet légitime

Chargement de l'applet

- Il faut disposer des clés de chargement de la carte
- Ces clés sont distribuées de manière sécurisée

Contre-mesures

Contre-mesures possibles

- Etendre les méta-données aux types natifs
- Utilisation de la segmentation mémoire matérielle

Contre-mesure mise en place

- Recommendations sécuritaires dans les guides d'utilisation
  - Vérifier systématiquement le code des applets
  - N'autoriser le chargement d'applets qu'aux entités de confiance

Des questions ?