Sécurité des Plateformes Java Card

Attaques et Contremesures

Guillaume Barbu
Crypto & Security Lab

IDEMIA

December 8, 2017

Outline

Introduction

1

Objectifs
Les notions de ”sécurité Java Card”

Objectifs de la journée

Présenter les problématiques de sécurité Java Card

- Sécurité de la plateforme (JCRE)
- Sécurité des applets

Objectifs de la journée

Présenter les problématiques de sécurité Java Card

- **Sécurité de la plateforme (JCRE)**
- Sécurité des applets

Objectifs de la journée

Présenter les problématiques de sécurité Java Card

- **Sécurité de la plateforme (JCRE)**
- Sécurité des applets

Présenter quelques attaques et contremesures

- De manière un peu théorique (ce matin)
- En pratique (cet après-midi)

Objectifs de la journée

Présenter les problématiques de sécurité Java Card

- **Sécurité de la plateforme (JCRE)**
- Sécurité des applets

Présenter quelques attaques et contremesures

- De manière un peu théorique (ce matin)
- En pratique (cet après-midi)

Prouver qu'il y a vraiment des gens dont c'est le travail

Besoin d'une plateforme sécurisée?

Utilisation des plateformes Java Card

- Bancaire
- Identité
- Télécom
- Pay-TV
- eSE

Besoin d'une plateforme sécurisée?

Utilisation des plateformes Java Card

- Bancaire
- Identité
- Télécom
- Pay-TV
- eSE

Dans tous ces domaines des *données* sensibles sont stockées sur la carte et utilisées par les applications embarquées

Quelles données sensibles?

En pratique ces *données* peuvent être:

- Des objets critiques stockés (chiffrés en général) dans la NVM de la carte: OwnerPIN, Key, ...
- Des données utilisées par les applications: machine d'état, bit de configuration, compteurs divers
- Le code des applications (stocké en NVM)
- Le flot d'exécution des applications ($\sim$ le code, mais à l'exécution cette fois)

Quelles données sensibles?

En pratique ces *données* peuvent être:

- Des objets critiques stockés (chiffrés en général) dans la NVM de la carte: OwnerPIN, Key, ...
- Des données utilisées par les applications: machine d'état, bit de configuration, compteurs divers
- Le code des applications (stocké en NVM)
- Le flot d'exécution des applications ($\sim$ le code, mais à l'exécution cette fois)

En fait ces *données* sont souvent plus sensibles du point de vue du fournisseur de service que de celui de l'utilisateur :-)

Quelles données sensibles?

En pratique ces *données* peuvent être:

- Des objets critiques stockés (chiffrés en général) dans la NVM de la carte: OwnerPIN, Key, ...
- Des données utilisées par les applications: machine d'état, bit de configuration, compteurs divers
- Le code des applications (stocké en NVM)
- Le flot d'exécution des applications ($\sim$ le code, mais à l'exécution cette fois)

En fait ces *données* sont souvent plus sensibles du point de vue du fournisseur de service que de celui de l'utilisateur :-)

C'est en partie ce qui motive le fait de considérer que l'attaquant a la main sur la carte et peut éventuellement installer sa propre applet pour essayer d'attaquer

Quelles menaces prendre en compte?

Attaques HW dites passive

- Espionner la communication entre la carte et le lecteur (Man in the middle)
- Analyser le temps d'exécution d'une application
- Analyser la consommation de courant de la carte pendant l'exécution d'une application
- Analyser les émanations eléctromagnétiques de la carte pendant l'exécution d'une application

Quelles menaces prendre en compte?

Attaques HW dites active

Différents moyens peuvent être utilisés pour perturber le fonctionnement du composant et induire des erreurs de *calcul*:

- LASER
- Champ eléctromagnétique
- Aiguille soumise à un pic de tension

Ces erreurs peuvent impacter les cellules de mémoire, les bus, les processeurs (CPU/Crypto)

Quelles menaces prendre en compte?

Attaques SW

Dans le cadre de l'analyse de la sécurité d'une plateforme on considère généralement que l'attaquant peut charger sa propre application

Cette application peut être *valide* ou *mal formée* (l'attaquant ne va pas forcément passer le BCV)

Les différents types de contremesures

Différentes attaques -> différentes contremesures

Attaque passive Généralement basées sur l'introduction de données aléatoires

Attaque active Généralement basées sur de la redondance (donnée/calcul)

Attaque soft La vérification est obligatoire. Le firewall assure l'isolation inter-application

Dans le cas des attaques HW, le composant lui-même intègre généralement des contremesures particulières (scrambling de mémoire, détecteur/capteur divers)

La carte peut se suicider lorsque de telles attaques sont détectées

Les Attaques SW et Combinées

2

Applet mal formée
Injection de Faute et Attaque Combinée

Applet mal formée

Qu'est ce qu'une applet mal formée?

C'est une applet qui ne respecte pas:

- le format standard d'un fichier .Cap
- les règles du langage Java Card
- les versions des fichiers .Exp

Applet mal formée

Qu'est ce qu'une applet mal formée?

C'est une applet qui ne respecte pas:

- le format standard d'un fichier .Cap
- les règles du langage Java Card
- les versions des fichiers .Exp

.Cap: Cela peut être des liens supprimés, corrompus ou non-résolus au chargement

Langage: Cela peut être l'utilisation de séquence de bytecodes non compatibles

.Exp: Cela peut être l'utilisation dans une applet d'une version différente de celle déclarée au chargement de l'applet sur la carte

Applet mal formée: Pour quoi faire?

Lorsqu'une applet suit le process *normal* de génération:

$$\text{.JAVA} \Rightarrow \text{.CLASS} \Rightarrow \text{.CAP}$$

On est sûr que le .CAP généré respecte le format défini par la spécification ainsi que les règles du langages Java

En particulier en Java les opérations arithmétiques sur les pointeurs ne sont pas autorisées:

```
short * a_ptr;
short an_offset;
short * another_ptr = a_ptr + an_offset;
```

En C cela fonctionne parfaitement

Applet mal formée: Pour quoi faire?

Lorsqu'une applet suit le process *normal* de génération:

$$\text{.JAVA} \Rightarrow \text{.CLASS} \Rightarrow \text{.CAP}$$

On est sûr que le .CAP généré respecte le format défini par la spécification ainsi que les règles du langages Java

En particulier en Java les opérations arithmétiques sur les pointeurs ne sont pas autorisées:

```
Object an_obj;
short an_offset;
Object another_obj = an_obj + an_offset;
```

Applet mal formée: Pour quoi faire?

Lorsqu'une applet suit le process *normal* de génération:

$$\text{.JAVA} \Rightarrow \text{.CLASS} \Rightarrow \text{.CAP}$$

On est sûr que le .CAP généré respecte le format défini par la spécification ainsi que les règles du langages Java

En particulier en Java les opérations arithmétiques sur les pointeurs ne sont pas autorisées:

```
Object an_obj;
short an_offset;
Object another_obj = an_obj + an_offset;
```

En Java ça ne compile pas...

Applet mal formée: Pour quoi faire?

Lorsqu'une applet suit le process *normal* de génération:

$$\text{.JAVA} \Rightarrow \text{.CLASS} \Rightarrow \text{.CAP}$$

On est sûr que le .CAP généré respecte le format défini par la spécification ainsi que les règles du langages Java

En particulier en Java les opérations arithmétiques sur les pointeurs ne sont pas autorisées:

```
Object an_obj;
short an_offset;
Object another_obj = an_obj + an_offset;
```

Cependant en modifiant après coup le CAP (ou le CLASS) tout devient possible!

Modification du code binaire

Le compilateur refuse de traiter l'addition `an_obj + an_offset` ?

Mais rien ne nous empêche de le faire à sa place!

On génère le code correspondant à

```
Object an_obj;
short an_offset;
Object another_obj = an_obj;
an_offset = 19 + an_offset;
```

Et analysons un peu le bytecode généré

Modification du code binaire

On obtient le bytecode suivant

```
aload_1    // on empile la référence de la variable locale <1> an_obj
           // sur la pile d'opérande
astore_3   // on dépile le sommet de la pile d'opérande et on stocke
           // le résultat dans la variable locale <3> another_obj
sspush 19  // on empile la valeur 19 sur la pile d'opérande
sload_2    // on empile la valeur de la variable locale <2> an_offset
           // sur la pile d'opérande
sadd       // on effectue l'addition des 2 éléments au sommet de la
           // pile d'opérande (en les dépilant) et on empile le résultat
sstore_2   // on dépile le sommet de la pile d'opérande et on stocke
           // le résultat dans la variable locale <2> an_offset
```

Modification du code binaire

Qu'il est possible de modifier

```
aload_1    // on empile la référence de la variable locale <1> an_obj
           // sur la pile d'opérande
NOP        // no operation
NOP        // no operation
sload_2    // on empile la valeur de la variable locale <2> an_offset
           // sur la pile d'opérande
sadd       // on effectue l'addition des 2 éléments au sommet de la
           // pile d'opérande (en les dépilant) et on empile le résultat
ASTORE_3   // on dépile le sommet de la pile d'opérande et on stocke
           // le résultat dans la variable locale <3> another_obj
```

Applet mal formée: Pour quoi faire?

Un autre exemple de corruption de fichier CAP utilisé dans la littérature consiste à éviter que certains liens soient résolus par l'*installer* lorsque l'applet est chargée sur la carte

getstatic et putstatic

- Les instructions getstatic et putstatic permettent de lire/écrire un champ static
- Fixer l'adresse de ce champ revient à obtenir un pointeur à une adresse arbitraire
- Les champs static ne sont pas protégés par l'isolation inter-application

invokestatic

- L'instructions invokestatic permet d'appeler une méthode static
- Fixer l'adresse de ce champ revient à obtenir un "pointeur de méthode" sur une adresse arbitraire

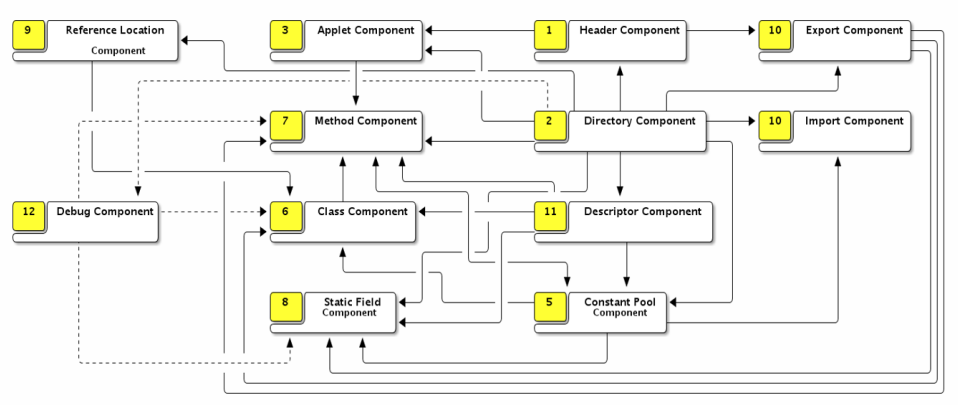
Modification des informations de résolution de lien

Cependant ce genre d'attaque requiert plusieurs étapes.
Dans le code généré, les paramètres des instructions getstatic, putstatic et invokestatic sont des indices renvoyant à un autre composant du .CAP: le Reference Location Component

Afin de fixer une valeur pour cette indice et que le linker ne tente pas de résoudre ce lien au chargement il faut donc également supprimer l'information du Reference Location Component

En résumé, une petite modification dans le Method Component peut entraîner de très nombreuses modifications dans le reste du CAP.

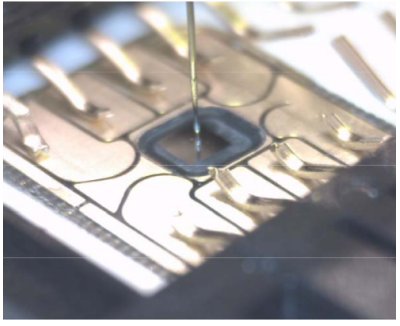
Interdépendance des composants du CAP



Injection de faute

Ces attaques consistent en général en l'apport d'énergie:

- Au bon endroit
- Au bon moment
- Avec la bonne intensité
- Pendant la bonne période de temps



Attaques combinées

Un attaquant peut parvenir à contourner certains mécanismes de sécurité de la plateforme grâce à ce type d'attaque:

- Vérification du firewall inter-application
- Vérification contre des over/underflow

Mais cela peut également être utilisé pour transformer une applet valide (au chargement) en une applet mal-formée (à l'exécution)

- Saut d'une instruction checkcast
- Modification d'un index de variable locale

L'exemple type d'attaque combinée

Supposons que l'on ait deux classes différentes: A et B

Confusion de type sans applet mal-formée?

- A a; B b;
- a = b;

L'exemple type d'attaque combinée

Supposons que l'on ait deux classes différentes: A et B

Confusion de type sans applet mal-formée?

- A a; B b;
- a = b;
- a = (A) b;

L'exemple type d'attaque combinée

Supposons que l'on ait deux classes différentes: A et B

Confusion de type sans applet mal-formée?

- A a; B b;
- a = b;
- a = (A) b;
- a = (A) ((Object) b);

En effet on peut toujours caster un objet quelconque en instance de la classe Object

Et en toute généralité, le compilateur ne peut pas savoir si un cast redescendant la hierarchie de type est valide ou non

L'exemple type d'attaque combinée

Une instruction particulière est alors ajouté afin de vérifier la validité du cast à l'exécution:

checkcast

Cette instruction jette une exception si le cast est invalide et retourne *silencieusement* sinon.

Une injection de faute au moment précis où la décision de l'invalidité du cast (en l'occurence) est prise permet donc de : retourner silencieusement

La confusion de type est alors réalisée.

Mise en place de contremesures face à ces attaques

3

Contremesures *organisationnelles*
Contremesures logicielles
Réaction aux attaques

Gestion de contenu

CardManager

Le CardManager est une applet généralement pré-chargée sur la carte

C'est cette applet qui est autorisée à gérer le contenu de la carte et notamment les droits d'installation et de désinstallation d'applet sur la plateforme

Afin de charger une applet il est nécessaire de s'authentifier auprès du CardManager.

Les clés nécessaire à cette authentification ne sont pas données aux utilisateurs

BCV et vérification de signature

- *A priori* une organisation ne validera le chargement d'une applet que si celle-ci a été *vérifiée*
- Cette vérification est réalisée par le BCV
- Une fois le CAP vérifié, une signature est calculée (DAP) et sera vérifiée au chargement de l'applet afin d'éviter une corruption post-vérification

On parle souvent du BCV comme de la partie *off-card* du modèle de sécurité Java Card

Modèle de sécurité

Sécurité off-card

- Le BCV est exécutée off-card car le processus de vérification est assez lourd
- Le BCV vérifie la structure du CAP, la cohérence des types, vérifie la bonne utilisation de la pile, des variables locales, des champs d'instance/static, etc...
- Aujourd'hui certaines cartes supportent la vérification on-card

Sécurité on-card

- Le firewall inter-application
- Le mécanisme de transaction

Modèle de sécurité

Sécurité off-card

- Le BCV est exécutée off-card car le processus de vérification est assez lourd
- Le BCV vérifie la structure du CAP, la cohérence des types, vérifie la bonne utilisation de la pile, des variables locales, des champs d'instance/static, etc...
- Aujourd'hui certaines cartes supportent la vérification on-card

Sécurité on-card

- Le firewall inter-application
- Le mécanisme de transaction
- Les vérifications au runtime

Contremesures logicielles

Pourquoi des contremesures logicielles supplémentaires?

Contremesures logicielles

Pourquoi des contremesures logicielles supplémentaires?

- Cela semble une bonne idée, le BCV lui-même peut comporter des bugs

Contremesures logicielles

Pourquoi des contremesures logicielles supplémentaires?

- Cela semble une bonne idée, le BCV lui-même peut comporter des bugs
- Les attaques par injections de faute rendent le modèle de sécurité off-card un peu obsolètes

Contremesures logicielles

Pourquoi des contremesures logicielles supplémentaires?

- Cela semble une bonne idée, le BCV lui-même peut comporter des bugs
- Les attaques par injections de faute rendent le modèle de sécurité off-card un peu obsolètes
- La plateforme a besoin de travailler avec les applets afin de protéger les données sensibles de manière dynamique

Contremesures logicielles

Pourquoi des contremesures logicielles supplémentaires?

- Cela semble une bonne idée, le BCV lui-même peut comporter des bugs
- Les attaques par injections de faute rendent le modèle de sécurité off-card un peu obsolètes
- La plateforme a besoin de travailler avec les applets afin de protéger les données sensibles de manière dynamique

Par exemple l'utilisation d'une pile typée, de la redondance sur les mécanismes d'isolation inter-application, le chiffrement des différents assets avec des clés différentes

Contremesures logicielles

Pourquoi des contremesures logicielles supplémentaires?

- Cela semble une bonne idée, le BCV lui-même peut comporter des bugs
- Les attaques par injections de faute rendent le modèle de sécurité off-card un peu obsolètes
- La plateforme a besoin de travailler avec les applets afin de protéger les données sensibles de manière dynamique

Par exemple l'utilisation d'une pile typée, de la redondance sur les mécanismes d'isolation inter-application, le chiffrement des différents assets avec des clés différentes

Le compromis entre la sécurité et les faibles ressources disponibles sur une carte n'est pas toujours facile à trouver

Réaction aux attaques

0xDEAD

- Les attaques impliquant des injections de faute nécessitent en général beaucoup d'expérimentation
- Une fois les bons paramètres trouvés la reproductibilité de l'attaque est souvent assez faible: 10% $\rightarrow$ (^.^)
- Une politique de sécurité est mise en place au niveau de la carte pour décider que faire lorsqu'une attaque est détectée
- La réaction extrême est de tuer la carte dès la première erreur

Conclusion

4

Conclusion

Sécurité(s) Java Card

On s'est focalisé ici sur la sécurité de la JCVM, mais pour obtenir une applet sécurisée il est également nécessaire de considérer

- Le code des API Java Card: Util.arrayCopy / arrayCompare ...
- Le code de l'applet elle-même: sécurité des branchements conditionnels, comportement dépendant d'une variable sensible ...

Conclusion

Contremesure et politique de sécurité

On voit que la multiplicité des attaques rend la définition de contremesure générique difficile

On voit également la nécessité pour les industriels d'être au minimum à l'état de l'art de la sécurité

La gestion de risque est importante dans la définition de la politique de sécurité:

- on ne veut pas être attaqué
- mais on ne veut pas non plus que toutes les cartes se tuent sans raison sur le terrain

Questions?

IDEMIA

Join us on

www.idemia.com