

TP Sécurité Java Card

11 Décembre 2017

Introduction

Dans ce TP nous allons mettre en pratique les différents éléments relatifs à la notion de sécurité des cartes Java vu en cours. Le but est donc de manipuler et d'expérimenter un maximum, tout en conservant les cartes fonctionnelles ;-). Pour la structure de l'applet de test en elle-même, vous pouvez vous référer au code donné en Annexe A.

1 Confusion de type

Dans un premier nous allons essayer de créer une confusion de type dans le code exécuté par la carte. On a vu deux méthodes pour réaliser cette confusion de type :

- Par corruption de CAP file
- Par une mauvaise utilisation des interfaces partagées

À partir d'un CAP valide, effectuez les modification permettant de créer une confusion de type entre un tableau de byte et un tableau de short (et/ou inversement). Ceci peut être fait à partir des méthodes `getShortArray` et `getByteArray` en Annexe A.

Initialisez ensuite le tableau cible puis créez une commande permettant de lire le tableau retourné par `get(Byte/Short)Array`. Qu'observez-vous ? Pouvez-vous expliquer ce comportement ? Quel(s) autre(s) comportement(s) aurait-on pu attendre ?

Le corps des commandes suivantes se trouve en Annexe B :

- `INS_FILL_B_ARRAY`
- `INS_FILL_S_ARRAY`
- `INS_DUMP_B_ARRAY`
- `INS_DUMP_S_ARRAY`

Votre CAP corrompu passe-t-il le BCV ?

2 Représentation des objets

Nous allons maintenant chercher à déterminer comment les objets sont représentés dans la mémoire de la carte. Pour cela nous allons passer par une confusion de type.

Quelle confusion de type pourrait être utilisée ? Une possibilité est de réaliser une confusion de type entre un tableau de **byte** et une autre classe contenant des champs d'instances, par exemple de type **short** :

```
package fr.cryptis.security ;

public class Confusion {
    public short s1      = (short) 0x1111 ;
    protected short s2 = (short) 0x2222 ;
    private short s3     = (short) 0x3333 ;
    public short s4       = (short) 0x4444 ;
    protected short s5 = (short) 0x5555 ;
    private short s6      = (short) 0x6666 ;
}
```

Comment réaliser la confusion de type ? La confusion de type va avoir lieu si l'on parvient à utiliser une instance de la classe Confusion comme si c'était un tableau. Une approche tout à fait semblable à celle utilisée pour la confusion entre des tableaux de types différents peut être envisagée.

Implémentez cette confusion de type ainsi qu'une commande permettant d'obtenir un dump (d'une taille passée en paramètre) de l'objet *cible*.

Que pouvez-vous dire du dump obtenu ? Quelle est la taille du tableau de byte après confusion ? Il est conseillé d'incrémenter petit à petit la taille du dump demandé.

Tentez d'interpréter les dumps obtenus à partir d'instances de classes de l'API standard *intéressantes* telles que : AID, Applet, OwnerPIN, Cipher, AESKey (des initialisations de ces classes sont données en Annexe C si besoin)

En vous servant de la confusion de type inverse de la confusion tableau/Objet vue précédemment, analysez la structure des tableaux transients.

Quelles contremesures pourraient empêcher ce genre d'attaque ? Dans quelle mesure ?

3 Exploration de la frame d'exécution

Une autre étape de caractérisation peut consister à tenter de caractériser la structure de la frame d'exécution Java. La spécification décrit la frame comme la structure contenant les informations nécessaires à l'exécution d'une méthode. On peut donc s'attendre à une structure de la forme suivante :

Info ?		
↑ ?	Local Variables	↓ ?
↑ ?	Operand Stack	↓ ?

N.B. Pour chaque méthode, le nombre de variables locales et la taille maximum de la pile d'opérande sont connues au moment où la méthode est invoquée (ces informations sont présentes dans le `.CLASS Java`).

Quelles peuvent-être les données présentes dans la partie Info ?

Proposez des modifications de CAP susceptibles de permettre l'analyse de la structure de celle-ci.

Après avoir validé que ces modifications n'endommageront pas la carte, tentez de les exploiter.

Comment empêcher l'accès à la frame ? Comment empêcher des modifications de la frame ?

A L'applet de test

```
package fr.cryptis.security ;

import javacard.framework.* ;

public class TPSecuApplet extends Applet
{
    public static final byte INS_DUMP_S_ARRAY = (byte) 0x10 ;
    public static final byte INS_DUMP_B_ARRAY = (byte) 0x11 ;
    public static final byte INS_DUMP_CONFUSION_ARRAY = (byte) 0x12 ;
    public static final byte INS_DUMP_CONFUSION = (byte) 0x13 ;
    public static final byte INS_FILL_S_ARRAY = (byte) 0x80 ;
    public static final byte INS_FILL_B_ARRAY = (byte) 0x81 ;

    public Confusion confusion ;
    public byte[] bAConfusion ;
    public short[] sAConfusion ;
    public byte[] tbAReset1, tbAReset2, tbAReset3 ;
    public byte[] tbASelect1, tbASelect2, tbASelect3 ;

    public TPSecuApplet() {

        confusion = new Confusion() ;
        bAConfusion = new byte[31] ;
        sAConfusion = new short[31] ;

        tbAReset1 = JCSystem.makeTransientByteArray((short) 15,
                                                    JCSystem.CLEAR_ON_RESET) ;
        tbAReset2 = JCSystem.makeTransientByteArray((short) 16,
                                                    JCSystem.CLEAR_ON_RESET) ;
        tbAReset3 = JCSystem.makeTransientByteArray((short) 17,
                                                    JCSystem.CLEAR_ON_RESET) ;

        tbASelect1 = JCSystem.makeTransientByteArray((short) 15,
                                                    JCSystem.CLEAR_ON_DESELECT) ;
        tbASelect2 = JCSystem.makeTransientByteArray((short) 16,
                                                    JCSystem.CLEAR_ON_DESELECT) ;
        tbASelect3 = JCSystem.makeTransientByteArray((short) 17,
                                                    JCSystem.CLEAR_ON_DESELECT) ;
    }

    public static void install(byte array[], short offset, byte length)
        throws ISOException {
        TPSecuApplet applet = new TPSecuApplet() ;
        applet.register() ;
    }
}
```

```

public void process(APDU apdu) {

    if (selectingApplet()) { return; }

    byte[] buffer = apdu.getBuffer();
    short lc = 0;
    short le = 0;

    switch(buffer[ISO7816.OFFSET_INS])
    {
        // Le code des autres commandes est donné au cas par cas
        ...

        // Par défaut on envoie le SW indiquant que l'instruction
        // est inconnue
        default :
            ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }

}

}

public short[] getShortArray(byte[] bArray, short[] sArray)
{
    // Modification de CAP ici!
    return sArray;
}

public byte[] getByteArray(byte[] bArray, short[] sArray)
{
    // Modification de CAP ici!
    return bArray;
}

public static byte[] getByteArray(byte[] bArray,
                                   Confusion confusion)
{
    // Modification de CAP ici!
    return bArray;
}

public static Confusion getConfusion(byte[] bArray,
                                       Confusion confusion)
{
    // Modification de CAP ici!
    return confusion;
}

```

B Traitement de commande APDU pour la confusion de type

```
case INS_FILL_S_ARRAY :
{
    // On récupère les données à inscrire dans l'APDU
    lc = apdu.setIncomingAndReceive();

    // On vérifie que l'on ne dépasse pas la taille du tableau
    if ((short)(lc>>1) > sAConfusion.length)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    for (short i = 0; i < (lc>>1); i+=2)
        sAConfusion[(short) (i>>1)] =
            (short) ((buffer[(short) (ISO7816.OFFSET_CDATA + i)] << 8) +
                buffer[(short) (ISO7816.OFFSET_CDATA + i + 1)]);

    break;
}
case INS_FILL_B_ARRAY :
{
    // On récupère les données à inscrire dans l'APDU
    lc = apdu.setIncomingAndReceive();

    // On vérifie que l'on ne dépasse pas la taille du tableau
    if (lc > bAConfusion.length)
        ISOException.throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);

    Util.arrayCopy(buffer, ISO7816.OFFSET_CDATA, bAConfusion, (short) 0, lc);

    break;
}
case INS_DUMP_S_ARRAY :
{
    short[] l_sAConfusion = getShortArray(bAConfusion, sAConfusion);

    le = apdu.setOutgoing();

    for (short i = 0; i < (short) (le>>1); i++)
    {
        buffer[(short) (2*i)] = (byte) ((l_sAConfusion[i] >> 8) & 0xFF);
        buffer[(short) (2*i + 1)] = (byte) (l_sAConfusion[(short) (i+1)] & 0xFF);
    }

    apdu.sendBytes((short) 0, le);

    break;
}
```

```

}
case INS_DUMP_B_ARRAY:
{
    byte[] l_bAConfusion = getByteArray(bAConfusion, sAConfusion);

    le = apdu.setOutgoing();
    for (short i = 0; i < le; i++)
    {
        buffer[i] = l_bAConfusion[i];
    }

    apdu.sendBytes((short) 0, le);

    break;
}
case INS_DUMP_CONFUSION_ARRAY:
{
    byte[] l_bAConfusion = getByteArray(bAConfusion, confusion);

    // On récupère la taille du dump que l'on souhaite
    short dumpLength = Util.getShort(buffer, ISO7816.OFFSET_P1);

    // On ne passe pas par l'API standard pour la copie afin
    // d'éviter un éventuel contrôle de type
    for (short i = 0; i < dumpLength; i++)
    {
        buffer[i] = l_bAConfusion[i];
    }
    apdu.setOutgoingAndSend((short) 0, dumpLength);
    break;
}
case INS_DUMP_CONFUSION:
{
    Confusion l_confusion = getConfusion(bAConfusion, confusion);

    // On récupère la taille du dump que l'on souhaite
    short dumpLength = Util.getShort(buffer, ISO7816.OFFSET_P1);

    switch (dumpLength) {
        case (short) 8:
            Util.setShort(buffer, (short) 6, l_confusion.s5);
        case (short) 6:
            Util.setShort(buffer, (short) 4, l_confusion.s4);
        case (short) 4:
            Util.setShort(buffer, (short) 2, l_confusion.s2);
        case (short) 2:
            Util.setShort(buffer, (short) 0, l_confusion.s1);
    }
}

```

```

    apdu.setOutgoingAndSend((short) 0, dumpLength);
    break;
}

```

C Création d’instances de classes OwnerPIN, Cipher, AES-Key

```

OwnerPIN pin = new OwnerPIN(byte tryLimit, byte maxPINSize);
pin.update(buffer, ISO7816.OFFSET_CDATA, (byte) 6);
// On présente un mauvais PIN pour voir la différence
pin.check(buffer, ISO7816.OFFSET_CDATA, (byte) 6);
// Et on présente le bon PIN pour voir la différence
// On présente un mauvais PIN pour voir la différence
pin.check(buffer, ISO7816.OFFSET_CDATA, (byte) 6);

```

```

AESKey k = (AESKey) KeyBuilder.buildKey(KeyBuilder.TYPE_AES, KeyBuilder.LENGTH_AES_128);
k.setKey(buffer, ISO7816.OFFSET_CDATA);
// Voit-on la clé apparaître dans le dump?

```

```

Cipher cip = Cipher.getInstance(Cipher.ALG_AES_CBC_ISO9797_M2, false);
// Quelle différence voit-on entre avant et après l'init?
cip.init(k, Cipher.MODE_ENCRYPT);

```

D Table des bytecodes

TABLE 1 – Légende

dec	hex	bc
0	00	nop
1	01	aconst_null
2	02	sconst_m1
3	03	sconst_0
4	04	sconst_1
5	05	sconst_2
6	06	sconst_3
7	07	sconst_4
8	08	sconst_5
9	09	iconst_m1
10	0A	iconst_0
11	0B	iconst_1
12	0C	iconst_2
13	0D	iconst_3
14	0E	iconst_4
15	0F	iconst_5
16	10	bspush
17	11	ssppush
18	12	bipush
19	13	sipush
20	14	iipush
21	15	aload
22	16	sload
23	17	iload
24	18	aload_0
25	19	aload_1
26	1A	aload_2
27	1B	aload_3
28	1C	sload_0
29	1D	sload_1
30	1E	sload_2
31	1F	sload_3
32	20	iload_0
33	21	iload_1
34	22	iload_2
35	23	iload_3
36	24	aaload

dec	hex	bc
37	25	baload
38	26	saload
39	27	iaload
40	28	astore
41	29	sstore
42	2A	istore
43	2B	astore_0
44	2C	astore_1
45	2D	astore_2
46	2E	astore_3
47	2F	sstore_0
48	30	sstore_1
49	31	sstore_2
50	32	sstore_3
51	33	istore_0
52	34	istore_1
53	35	istore_2
54	36	istore_3
55	37	aastore
56	38	bastore
57	39	sastore
58	3A	iastore
59	3B	pop
60	3C	pop2
61	3D	dup
62	3E	dup2
63	3F	dup_x
64	40	swap_x
65	41	sadd
66	42	iadd
67	43	ssub
68	44	isub
69	45	smul
70	46	imul
71	47	sdiv
72	48	idiv
73	49	srem

dec	hex	bc
74	4A	irem
75	4B	sneg
76	4C	ineg
77	4D	sshl
78	4E	ishl
79	4F	sshr
80	50	ishr
81	51	sushr
82	52	iushr
83	53	sand
84	54	iand
85	55	sor
86	56	ior
87	57	sxor
88	58	ixor
89	59	sinc
90	5A	iinc
91	5B	s2b
92	5C	s2i
93	5D	i2b
94	5E	i2s
95	5F	icmp
96	60	ifeq
97	61	ifne
98	62	iflt
99	63	ifge
100	64	ifgt
101	65	ifle
102	66	ifnull
103	67	ifnonnull
104	68	if_acmpeq
105	69	if_acmpne
106	6A	if_scmpeq
107	6B	if_scmpne
108	6C	if_scmlt
109	6D	if_scmpge
110	6E	if_scmpgt

