TP Sécurité Java Card

3 février 2025

1 Première partie

1.1 L'applet de test

La trame de base de l'applet est à récupérer au début du TP.

Le constructeur du projet TP1 est rappelé ci-dessous. Le but des différentes manipulations réalisées au cours du TP sera de retrouver la valeur générée aléatoirement pour définir le PIN.

```
RandomData rd;

public TestAppletTP1() {
    // On va faire toutes les allocations de mémoire ici

    // On créé un tableau transient (ie. tableau en RAM, effacé dès que l'applet est déselectionnée)
    transientBA = JCSystem.makeTransientByteArray((short) 128, JCSystem.CLEAR_ON_DESELECT);

    // On créé des tableaux de byte et de short persistent
    bA = new byte[16];
    sA = new short[16];

    // On créé une instance de la classe C
    c = new C();

    // On génère 8 octets aléatoire pour initialiser notre PIN
    rd = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
    rd.generateData(transientBA, (short) 0, (short) 8);

    pin = new OwnerPIN((byte) 5, (byte) 8);
    pin.update(transientBA, (byte) 0, (byte) 8);

}
```

FIGURE 1 – Constructeur de l'applet TestAppletTP1

Cette applet supporte initialement 5 commandes APDU. Ces commandes suivent le format suivant :

CLA	INS	P1	P2	LC	DATA	LE

TABLE 1 – Format d'une commande APDU

La description des commandes est donnée ci-dessous :

INS_CHECK_PIN cette commande va tenter de valider le PIN avec les valeurs données dans le champs *DATA* de l'APDU. Si le PIN est correct, le *STATUS WORD* 9000 est renvoyé. Sinon le SW 66xx est renvoyé où xx indique le nombre d'essai restant pour valider le PIN (d'après l'initialisation du PIN ce nombre est initialement à 5).

INS_OBJECT_TO_SHORT cette commande renvoie la valeur retourner par la méthode *objectToShort*.

INS_SHORT_TO_C cette commande appelle la méthode *shortToC* en donnant en argument la valeur en P1|P2 de la commande APDU.

INS_SHORT_TO_ZSHORT cette commande appelle la méthode *shortToZShort* en donnant en argument la valeur en P1|P2 de la commande APDU.

INS_ZSHORT_TO_SHORT cette commande renvoie la valeur retourner par la méthode *zShortToShort*.

Récupérez le projet, importez le dans votre IDE puis lancez le.

Sélectionnez l'applet et essayez les différentes commandes. L'AID de l'applet à utiliser en paramètre de la commande /select est `66722E796F6D622E747001`. Quelle est la longueur de PIN attendu ? Voyez-vous le compteur d'essai se décrémenter quand vous tentez de valider le PIN ?

Le .CAP généré par l'IDE passe-t-il la vérification ? Le script verifycap.bat se trouve au même endroit que le projet. Il faut lui fournir les fichiers .EXP utilisé par l'applet (lang.exp, framework.exp et security.exp) en plus du .CAP à vérifier.

```
verifycap.bat ..\JC305\api_export_files\java\lang\javacard\lang.exp ..\JC305\
    api_export_files\javacard\framework\javacard\framework.exp ..\JC305\
    api_export_files\javacard\security\javacard\security.exp tp.cap
```

1.2 Corruption de .CAP

Ce matin nous avons vu qu'une façon d'attaquer une carte était de charger des fichiers .CAP modifiés.

À partir des méthodes *xToY*, avez-vous des corruptions de .CAP à proposer ? Afin de modifier le fichier .CAP vous pouvez vous servir de HxD [1] (ou d'un autre éditeur hexa).
Il faut dans un premier temps localiser le *Method Component* dans le fichier .CAP, puis identifier les opcodes à modifier. Pour cela, on peut chercher une séquence d'opcaode correspondant à ce qui est affiché dans la vue **Java Card Bytecode** de l'IDE (Fig. 2).

Si vous replacez le fichier modifié dans le répertoire bin de votre projet, que remarquez-vous dans la vue Java Card Bytecode de l'IDE ? Cela montre que vos modifications ont bien été prises en compte. Attention, toute modification de votre applet (suivi d'une sauvegarde) dans l'IDE entrainera l'écrasement de vos modifications (l'IDE regénère un nouveau .CAP automatiquement).

Que renvoie maintenant les commandes INS_X_TO_Y ?

Le .CAP passe-t-il toujours la vérification ? Indice :

1. `https://mh-nexus.de/en/hxd/`

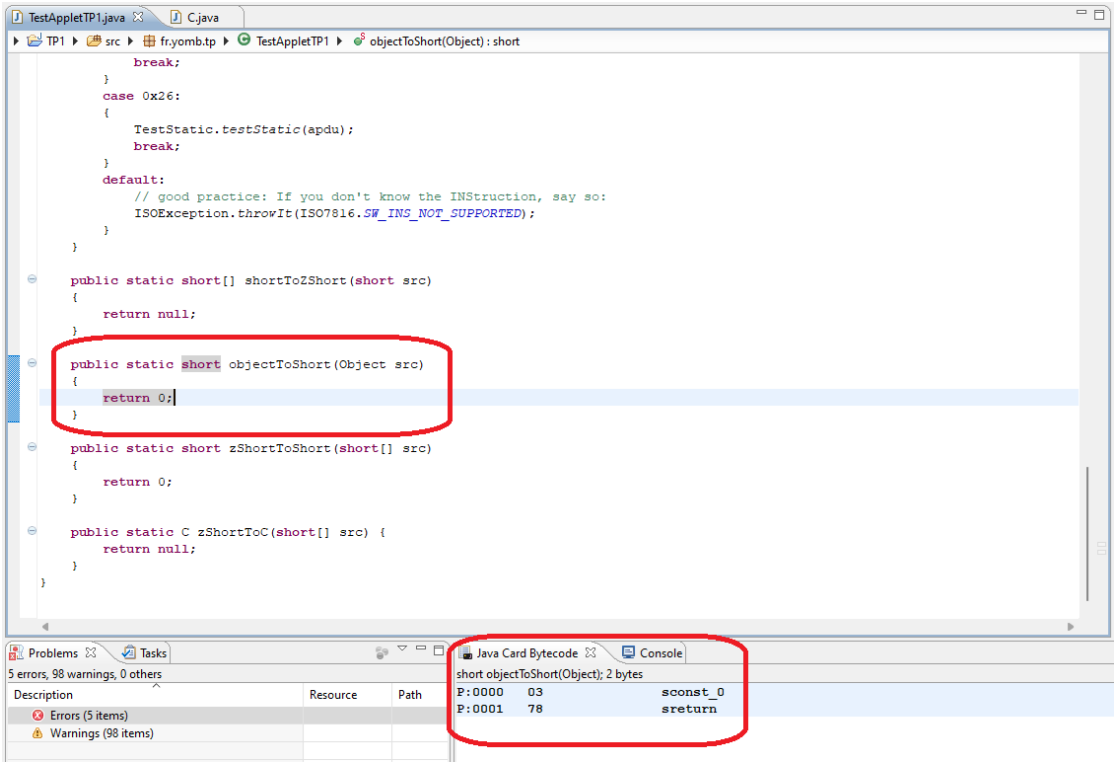

FIGURE 2 – Vue du bytecode de la méthode *objectToShort*

2 Deuxième partie

En plus de ne pas pouvoir passer l'étape de vérifiction, la modification de .CAP peut parfois être un peu fastidieuse. Dans le reste du TP nous passerons donc par un autre type d'attaque vu ce matin : l'abus des interfaces partagées.

Un second projet reprenant les modifications effectuées précédemment par ce biais est disponible en ligne.

Le comportement des commandes est-il bien le même ? Le .CAP passe-t-il la vérification ?

2.1 Analyse de la mécanique d'allocation mémoire

Exécutez les différentes commandes INS_X_TO_Y. Qu'observez vous sur les valeurs renvoyées par rapport à l'ordre dans lequel les objets sont créés dans le constructeur ? Afin d'affiner cette analyse on pourra procéder à la même expérience en modifiant la taille des tableaux sA et bA par exemple.

Est-il alors possible d'établir quelle est la référence de l'objet pin ciblé ? Indice :

Il est alors possible de lire le contenu de l'objet pin par la commande INS_SHORT_TO_ZSHORT. Pouvez-vous interpréter ce contenu ? Présenter un mauvais PIN pourra vous donner un peu d'information.

Comment retrouver le PIN ?

A Table des bytecodes

Table 2 – 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	sspush
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_scmplt
109	6D	if_scmpge
110	6E	if_scmpgt

dec	hex	mnemonic
111	6F	if_scmple
112	70	goto
113	71	jsr
114	72	ret
115	73	stableswitch
116	74	itableswitch
117	75	slookupswitch
118	76	ilookupswitch
119	77	areturn
120	78	sreturn
121	79	ireturn
122	7A	return
123	7B	getstatic_a
124	7C	getstatic_b
125	7D	getstatic_s
126	7E	getstatic_i
127	7F	putstatic_a
128	80	putstatic_b
129	81	putstatic_s
130	82	putstatic_i
131	83	getfield_a
132	84	getfield_b
133	85	getfield_s
134	86	getfield_i
135	87	putfield_a
136	88	putfield_b
137	89	putfield_s
138	8A	putfield_i
139	8B	invokevirtual
140	8C	invokespecial
141	8D	invokestatic
142	8E	invokeinterface
143	8F	new
144	90	newarray
145	91	anewarray
146	92	arraylength
147	93	athrow

dec	hex	mnemonic
148	94	checkcast
149	95	instanceof
150	96	sinc_w
151	97	iinc_w
152	98	ifeq_w
153	99	ifne_w
154	9A	iflt_w
155	9B	ifge_w
156	9C	ifgt_w
157	9D	ifle_w
158	9E	ifnull_w
159	9F	ifnonnull_w
160	A0	if_acmpeq_w
161	A1	if_acmpne_w
162	A2	if_scmpeq_w
163	A3	if_scmpne_w
164	A4	if_scmplt_w
165	A5	if_scmpge_w
166	A6	if_scmpgt_w
167	A7	if_scmple_w
168	A8	goto_w
169	A9	getfield_a_w
170	AA	getfield_b_w
171	AB	getfield_s_w
172	AC	getfield_i_w
173	AD	getfield_a_this
174	AE	getfield_b_this
175	AF	getfield_s_this
176	B0	getfield_i_this
177	B1	putfield_a_w
178	B2	putfield_b_w
179	B3	putfield_s_w
180	B4	putfield_i_w
181	B5	putfield_a_this
182	B6	putfield_b_this
183	B7	putfield_s_this
184	B8	putfield_i_this

dec	hex	mnemonic
254	FE	impdep1
255	FF	impdep2