Numéro d'anonymat :

Partie J. Lancia

4 points
Répondre sur le sujet !

Questions de cours

plusieurs réponses possibles pour chaque question
chaque mauvaise réponse retranche des dixièmes de points

1. La machine virtuelle Java Card est …

☐ Une machine à pile
☐ Un programme Java
☐ Un processeur simulé
☐ Un système d'exploitation

2. Cycle de vie d'une applet

☐ Les applets java sont compilées vers un fichier class
☐ Le fichier exp permet d'exporter des fonctions
☐ Le fichier jca est indispensable pour charger une applet
☐ Le fichier cap contient le code des méthodes de l'applet

3. Une applet peut être est chargée …

☐ Dans un Security Domain
☐ Sans authentification cryptographique
☐ Sans être vérifiée par le BCV
☐ Sous forme de fichier jar

4. Quel mécanisme intégré dans la carte permet d'assurer qu'une applet a été validée ?

☐ Le DAP
☐ La vérification de Token
☐ Le BCV
☐ Le firewall

5. L'interface Shareable de l'API Java Card…

☐ Permet le contournement du firewall
☐ Déclare des méthodes partagées
☐ Permet le partage d'objet
☐ Permet les attaques en stack overflow

6. Allocation mémoire

☐ Les objets sont alloués en mémoire persistante
☐ Tous les tableaux sont alloués en mémoire persistante
☐ Les variables locales sont allouées en mémoire transiente
☐ Toutes les références sont allouées en mémoire persistante

7. Pile et frame

☐ Les locales sont stockées dans la pile d'opérande
☐ La frame a une taille constante pour une méthode données
☐ Les arguments d'une fonction sont stockés dans les locales
☐ La pile est systématiquement typée

8. Les écritures mémoires dans une transaction

☐ Sont toutes systématiquement réalisées
☐ Sont réalisées dans leur intégralité ou pas du tout
☐ Peuvent être annulées
☐ Provoquent des débits et des crédits

9. Le contexte d'exécution

☐ Détermine les droits d'accès aux objets
☐ Ne change jamais pendant l'exécution d'une applet
☐ Détermine le possesseur d'un objet lors de sa création
☐ Est identique pour chaque package

10. Les bytecodes

☐ Sont typés
☐ Sont interprétés par le micro-processeur
☐ Agissent sur la pile d'opérande
☐ Agissent sur les locales

Questions pratiques

```
.method private method1(Ljava/lang/Object;)S {
    .stack 1;
    .locals 1;

    .descriptor   Ljava/lang/Object;      1.0;

        L0: sconst_0;
            sstore_2;
            aload_1;
            sstore_2;
            sload_2;
            sreturn;
}

.method private method2()V {
    .stack 1;
    .locals 4;

        L0: sspush 4369;
            sstore_1;
            sspush 8738;
            sstore_2;
            sspush 13107;
            sstore_3;
            sspush 17476;
            sstore 4;
            sspush 4369;
            sspush 4369;
            sstore_1;
            return;
}
```

```
.method private method3()V {
    .stack 1;
    .locals 4;

        L0: sspush 4369;
            sstore_1;
            sspush 8738;
            sstore_2;
            sspush 13107;
            sstore_3;
            sspush 17476;
            sstore 4;
            pop;
            return;
}
```

1. Quelle méthode réalise une attaque de type stack overflow

☐ method1
☐ method2
☐ method3

2. Quelle méthode réalise une attaque de type stack underflow

☐ method1
☐ method2
☐ method3

3. Quelle méthode réalise une attaque de type confusion de type

☐ method1
☐ method2
☐ method3

4. Supprimez une instruction dans la méthode 4 pour produire une confusion de type (barrez l'instruction).

```
.method private method4(Ljava/lang/Object;)[S {
    .stack 1;
    .locals 0;

    .descriptor     Ljava/lang/Object;      1.0;

    L0: sspush 4369;
        sstore_1;
        sspush 8738;
        sstore_2;
        sspush 13107;
        sstore_3;
        sspush 17476;
        sstore 4;
        aload_1;
        checkcast 12 0;
        areturn;
}
```