TD DevSecEmb

TD sécurité et systeme embarqué

TD1: Emily voici les différentes étapes réaliser pour patch le binaire:

```
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$ gcc program.c -o program
program.c: In function ‘is_valid’:
program.c:5:6: warning: implicit declaration of function ‘strcmp’ [-Wimplicit-function-declarat
ion]
  if (strcmp(password, "poop") == 0){
      ^~~~~~
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$ ./program
please input a word: butts
that's not correct !
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$ ./program
please input a word: poop
that's correct !
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$
```

```
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$ file program
program: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
 for GNU/Linux 3.2.0, BuildID[sha1]=559f1bbcb63e0f86437c351fe40304e07496f558, not stripped
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$ hexdump -C program | head -n 20
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 3e 00 01 00 00 00  a0 10 00 00 00 00 00 00  |..>.............|
00000020  40 00 00 00 00 00 00 00  88 3a 00 00 00 00 00 00  |@........:......|
00000030  00 00 00 00 40 00 38 00  0b 00 40 00 1e 00 1d 00  |....@.8...@.....|
00000040  06 00 00 00 04 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000050  40 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |@.......@.......|
00000060  68 02 00 00 00 00 00 00  68 02 00 00 00 00 00 00  |h.......h.......|
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  |................|
00000080  a8 02 00 00 00 00 00 00  a8 02 00 00 00 00 00 00  |................|
00000090  a8 02 00 00 00 00 00 00  1c 00 00 00 00 00 00 00  |................|
000000a0  1c 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  |................|
000000b0  01 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  a0 06 00 00 00 00 00 00  a0 06 00 00 00 00 00 00  |................|
000000e0  00 10 00 00 00 00 00 00  01 00 00 00 05 00 00 00  |................|
000000f0  00 10 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
00000100  00 10 00 00 00 00 00 00  ad 02 00 00 00 00 00 00  |................|
00000110  ad 02 00 00 00 00 00 00  00 10 00 00 00 00 00 00  |................|
00000120  01 00 00 00 04 00 00 00  00 20 00 00 00 00 00 00  |......... ......|
00000130  00 20 00 00 00 00 00 00  00 20 00 00 00 00 00 00  |. ....... ......|
user@localhost:~/Documents/TD_Securite_SysEmbarquer/program$
```

```
0000000000001185 <is_valid> (Offset dans le fichier : 0x1185):
is_valid():
    1185:       55                      push   %rbp
    1186:       48 89 e5                mov    %rsp,%rbp
    1189:       48 83 ec 10             sub    $0x10,%rsp
    118d:       48 89 7d f8             mov    %rdi,-0x8(%rbp)
    1191:       48 8b 45 f8             mov    -0x8(%rbp),%rax
    1195:       48 8d 35 68 0e 00 00    lea    0xe68(%rip),%rsi        # 2004 <_IO_stdin_used+0x4> (Offset dans le fichier : 0
x2004)
    119c:       48 89 c7                mov    %rax,%rdi
    119f:       e8 bc fe ff ff          callq  1060 <strcmp@plt> (Offset dans le fichier : 0x1060)
    11a4:       85 c0                   test   %eax,%eax
    11a6:       75 07                   jne    11af <is_valid+0x2a> (Offset dans le fichier : 0x11af)
    11a8:       b8 01 00 00 00          mov    $0x1,%eax
    11ad:       eb 05                   jmp    11b4 <is_valid+0x2f> (Offset dans le fichier : 0x11b4)
    11af:       b8 00 00 00 00          mov    $0x0,%eax
    11b4:       c9                      leaveq
    11b5:       c3                      retq
```

```
00001170  c3 0f 1f 80 00 00 00 00  c3 0f 1f 80 00 00 00 00  |................|
00001180  e9 7b ff ff ff 55 48 89  e5 48 83 ec 10 48 89 7d  |.{...UH..H...H.}|
00001190  f8 48 8b 45 f8 48 8d 35  68 0e 00 00 48 89 c7 e8  |.H.E.H.5h...H...|
000011a0  bc fe ff ff 85 c0 75 07  b8 01 00 00 00 eb 05 b8  |......u.........|
000011b0  00 00 00 00 c9 c3 55 48  89 e5 48 83 ec 10 48 c7  |......UH..H...H.|
000011c0  45 f8 00 00 00 00 bf 00  01 00 00 e8 a0 fe ff ff  |E...............|
000011d0  48 89 45 f8 48 8d 3d 2e  0e 00 00 b8 00 00 00 00  |H.E.H.=.........|
```

```
user@localhost:~/Documents/TD_Securite_SysEmbarquer/TD/TD2/program$ ./program
please input a word: any
that's correct !
```

Réponses aux questions:

1. La principale différence entre les deux architectures tient dans les choix de conception de leur jeu d'instruction : l'architecture x86 est une architecture

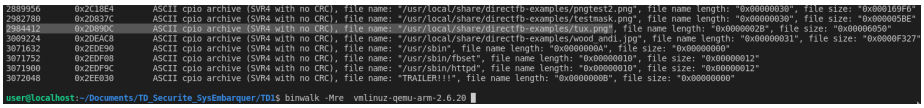
CISC (Complex Instruction Set Computer), tandis que l'ARM est une architecture RISC (Reduced ISC).

Cela signifie que les puces ARM ne supportent que des instructions simples et de taille fixe (4 octets pour le jeu d'instructions ARM standard, 2 octets pour le jeu réduit Thumb), s'exécutant en un nombre constant de cycles. A l'inverse les puces x86, permettent de réaliser certaines tâches complexes car les instructions nécessitent plus de cycle.

2. avec le langage assembleur on remarque qu'une boucle for s'execute grâce à des jump entre différentes addresse. On peut donc se servir de ce système pour modifiez l'addresse d'arriver du jump pour executer une autre instruction par exemple.

3. Le patching est par définition difficile à empecher, le seul moyen d'essayer de dissuader des attaquants serait de rendre le code le plus complexe possible au niveau assembleur. Pour cela il suffit de rajouter du bruit dans le code, c'est à dire de rajouter des portions de codes inutiles qui ne servent qu'à mélanger du vrai code dans du faux afin de dissuader des attaquants.

TD2 Binwalk On utilise binwalk et en cherchant dans tous les fichier utiliser par l'image on cherche l'image du pingouin de linux. Après des recherches rapide on peut savoir que le pingouin de linux se nomme Tux. On fait donc une recherche sur **"tux"** et on trouve sa location dans l'image assez rapidement:



Réponse aux questions

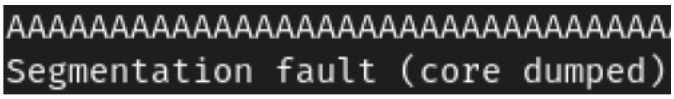
1. Même si on trouve rapidement l'image de tux assez rapidement, on se rend compte que le modifier n'est pas si simple. Ce qui nous en empêche, doit être le fait qu'on ne possède pas les d'écriture sur l'image

2. Pour contourner cette situation il faut réussir à démarrer l'image linux en tant que root et non en tant qu'user.

3.

TD3 Speedrun Une fois le build executé et l'image d'ubuntu executé on lance le binaire ***speedrun-001*** en étant root et le programme nous propose d'entrer une chaine de caractère.

On entre alors une chaine de caractère assez longue pour dépasser la mêmoire et créer un buffer overflow. on rentre donc une suite assez longue de A et on obtient:



on peut maintenant utiliser l'outils gdb afin de connaitre l'offset du dépassement afin de savoir ou ajouter du code.

une fois l'offset trouvé on utilise une rop chain afin de pouvoir injecter un buffer assez long et de modifier l'execution du binaire. On obitent alors:

Question:

1. Une fois root, un attaquant bénéficie des droits administrateur et peux donc executer tous ce qu'il veut dans le système. Ainsi avec les droits root la seule limite d'un attaquant serait son imagination.
2. Tout d'abord il faut absolument éviter d'avoir un code qui permet le segmentation fault. Il faut tenter de proteger son code des bugs en les anticipants le plus possible. Si on peut anticiper un bug alors il est possible, lors du développement du programme, de gérer ces bugs, par exemple à l'aide de try catch. Cela peut permettre de diminuer la surface d'attaque.

TD4 Toolbox Voici des liens vers d'autres github décrivant des attaques man in the middle sur différentes surface. Pour des raisons de manques de moyens matériels, je n'ai malheuresement pas pu tester ces méthodes, toutefois les analyser et comprendre leur principe fut très enrichissant et m'a permis de mieux appréhender ces interfaces/communication. J'espère qu'elles seront utiles pour la fabrication de votre toolBox

Attaques Man in the Middle sur:

-USB: https://github.com/0x7ace80/virtualbox_usb_mitm

-Bluetooth: https://github.com/DigitalSecurity/btlejuice

-WiFi: https://github.com/maneshthankappan/Multi-Channel-Man-in-the-Middle-Attacks-Against-Protected-Wi-Fi-Networks-By-Improved-Variant

-NFC: https://github.com/EMVrace/EMVrace.github.io/blob/master/README.md

Question

1. Pour évaluer la criticité d'une vlnérabilité on va prendre deux paramètre en compte:
   - Sa probabilité de survenir
   - le type et la quantité de données en danger

En effet si la probabilté qu'une faille soit exploiter est grande mais que les données impacté ne sont pas importante ou en très faible quantité, alors la vulnérabilité ne sera pas critique. Il en est de même si les données impactées sont en grande quantité ou très importantes mais que la probabilité que la faille soit exploité est extrememement faible. A l'inverse si la probabilité est élévé et que les données sont importantes alors la vulnérabilité peut être considérer comme critique. Tout ceci peut être résumé dans ce tableau évaluant la criticité d'un vulnérabilité:

IMPACT	IMPORTANT (7 à 9)	Moyenne	Importante	Critique
	MOYEN (4 à 6)	Faible	Moyenne	Importante
	FAIBLE (1 à 3)	Information	Faible	Moyenne
		FAIBLE (1 à 3)	MOYENNE (4 à 6)	IMPORTANTE (7 à 9)
		PROBABILITE		