

TD DevSecEmb

TD sécurité et système 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-declaration]
    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 00 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 00 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 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$
```

```
000000000001185 <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     0xe8(%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  %eax
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 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'exécute grâce à des jump entre différentes adresse. On peut donc se servir de ce système pour modifier l'adresse d'arriver du jump pour exécuter une autre instruction par exemple.
3. Le patching est par définition difficile à empêcher, 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:

```
288956 0x2C18E4 ASCII cpio archive (SVR4 with no CRC), file name: "/usr/local/share/directfb-examples/pingtest2.png", file name length: "0x00000020", file size: "0x000109F0"
288288 0x2D337C ASCII cpio archive (SVR4 with no CRC), file name: "/usr/local/share/directfb-examples/testimg2.png", file name length: "0x00000020", file size: "0x000005B0"
2884412 0x2D080C ASCII cpio archive (SVR4 with no CRC), file name: "/usr/local/share/directfb-examples/tux.png", file name length: "0x00000028", file size: "0x00000030"
2889224 0x2DEAC8 ASCII cpio archive (SVR4 with no CRC), file name: "/usr/local/share/directfb-examples/wood_endi.jpg", file name length: "0x00000031", file size: "0x0000F327"
3871632 0x2EDF98 ASCII cpio archive (SVR4 with no CRC), file name: "/usr/sbin", file name length: "0x00000004", file size: "0x00000000"
3871752 0x2EDF98 ASCII cpio archive (SVR4 with no CRC), file name: "/usr/sbin/tbset", file name length: "0x00000010", file size: "0x00000012"
3871800 0x2EDF9C ASCII cpio archive (SVR4 with no CRC), file name: "/usr/sbin/tbset", file name length: "0x00000010", file size: "0x00000012"
3872048 0x2EDF98 ASCII cpio archive (SVR4 with no CRC), file name: "TRAILER!!!", file name length: "0x00000008", file size: "0x00000000"
user@localhost:~/Documents/TD_Securite SysEmbarquer/TD15 binwalk -Mr vmlinux-gnu-arm-2.6.20
```

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 mémoire et créer un buffer overflow. on rentre donc une suite assez longue de A et on obtient:

AAAAAAAAAAAAAAAAAAAAAAAAAAAA
Segmentation fault (core dumped)

on peut maintenant utiliser l'outil gdb afin de connaître l'offset du dépassement afin de savoir où 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'exécution du binaire. On obtient alors:

```
root@ca7e702d0b7c:/home# (cat rop; cat) | ./speedrun-001
Hello brave new challenger
Any last words?
This will be the last thing that you say: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9A
f6A7f7f8f9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al
6An7Am8Am9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At
At7AT8AT9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az
a7Ba8Ba9B0B1B2B3B4B5B6B7B8B9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9BdBd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg
7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi6Bi7Bi8Bi9Bj0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9BkBk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9Bl0Bl1Bl2Bl3Bl4Bl5Bl6Bl7Bl8Bl9Bm0Bm1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn
uid=0(root) gid=0(root) groups=0(root)
```

Question:

1. Une fois root, un attaquant bénéficie des droits administrateur et peut donc exécuter tout 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 protéger son code des bugs en les anticipant 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 surfaces. Pour des raisons de manque de moyens matériels, je n'ai malheureusement 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 vulnérabilité on va prendre deux paramètres en compte:
 - Sa probabilité de survenir
 - le type et la quantité de données en danger

En effet si la probabilité qu'une faille soit exploitée est grande mais que les données impactées ne sont pas importantes 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 exploitée est extrêmement faible. A l'inverse si la probabilité est élevée et que les données sont importantes alors la vulnérabilité peut être considérée comme critique. Tout ceci peut être résumé dans ce tableau évaluant la criticité d'une 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		