AmarOk

Zeste de savoir

r2, pour changer de gdb

21 janvier 2019

Table des matières

% R2, POUR CHANGER DE GDB % AmarOk % 24 mars 2018

1. Introduction

Hei, je n'avais pas n'avais pas écrit de billet ici alors que les sujets ne manquent pas...

Aujourd'hui avec des personnes de ma boîte, nous allons peut-être monter une équipe pour le NorthSec 2018 à Montréal. Il m'arrive très rarement de faire des CTF et je n'en ai pas fait depuis un moment.

Pour se préparer, on va se faire une séance de quelques heures par semaine pour faire quelques challenges. Et j'ai décidé d'apprendre à utiliser `radare2` (http ://radare.org/). Dans ce billet, je vais donc partager mon expérimentation du jour, ou des suivantes si j'en fais une série.

Pour symboliser cette décision, une des personnes a proposé un petit challenge, où, comme tous les CTF, il suffit de récupérer un flag. Voici comment j'ai utilisé `r2` :

2. Première résolution, simple analyse.

Le challenge n'avait rien de compliqué et prenait quelques secondes avec `gdb` à résoudre, je ne vais donc pas en faire un write-up, juste me concentrer sur `r2`.

J'ai tout d'abord tenté une approche sans exécuter le programme, mais en regardant les fonctionnalités de reverse basiques.

La première étape est de lancer `r2` en lui passant le programme

```
1  AmarOk@tars3 ▶ ~/Downloads ▶ r2 -d ./easy_reverse
```

(le `-d` servira pour la seconde approche, par le debug)

Puis avec des tutoriels en ligne, je trouve les commandes `aa` pour lancer une analyse du programme (pour *anyalyze all*) puis (`pdf @main` pour obtenir le code qui m'intéresse). Ce qui donne :

```
[0x7ff4442a3ed0]> aa
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x7ff4442a3ed0]> pdf @main
            ;-- main:
/ (fcn) main 182
|   main ();
|           ; var int local_84h @ rbp-0x84
|           ; var int local_80h @ rbp-0x80
|           ; var int local_78h @ rbp-0x78
|           ; var int local_74h @ rbp-0x74
|           ; var int local_72h @ rbp-0x72
|           ; var int local_70h @ rbp-0x70
|           ; var int local_8h @ rbp-0x8
|             ; DATA XREF from 0x004005ad (entry0)
|           0x00400686      55              push rbp
|           0x00400687      4889e5          mov rbp, rsp
|           0x0040068a      4881ec900000.   sub rsp, 0x90
|           0x00400691      64488b042528.   mov rax, qword fs:[0x28]    ; [0x28:8]=-1 ; '(' ; 40
|           0x0040069a      488945f8        mov qword [local_8h], rax
|           0x0040069e      31c0            xor eax, eax
|           0x004006a0      48b8464c4147.   movabs rax, 0x3857397b47414c46
|           0x004006aa      48894580        mov qword [local_80h], rax
|           0x004006ae      c7458873626e.   mov dword [local_78h], 0x7a6e6273
|           0x004006b5      66c7458c387d    mov word [local_74h], 0x7d38
|           0x004006bb      c6458e00        mov byte [local_72h], 0
|           0x004006bf      bfc4074000      mov edi, str.Password_:     ; 0x4007c4 ; "Password : "
|           0x004006c4      b800000000      mov eax, 0
|           0x004006c9      e872feffff      call sym.imp.printf         ; int printf(const char *format)
|           0x004006ce      488d4590        lea rax, [local_70h]
|           0x004006d2      4889c6          mov rsi, rax
|           0x004006d5      bfd0074000      mov edi, 0x4007d0
|           0x004006da      b800000000      mov eax, 0
|           0x004006df      e88cfeffff      call sym.imp.__isoc99_scanf
|           0x004006e4      488d5580        lea rdx, [local_80h]
|           0x004006e8      488d4590        lea rax, [local_70h]
|           0x004006ec      4889d6          mov rsi, rdx
|           0x004006ef      4889c7          mov rdi, rax
|           0x004006f2      e869feffff      call sym.imp.strcmp         ; int strcmp(const char *s1, const char *s2)
|           0x004006f7      89857cffffff    mov dword [local_84h], eax
|           0x004006fd      83bd7cffffff.   cmp dword [local_84h], 0
|       ,=< 0x00400704      7511            jne 0x400717
|       |   0x00400706      bfd3074000      mov edi, str.You_re_a_Wizard ; 0x4007d3 ; "You're a Wizard"
|       |   0x0040070b      e810feffff      call sym.imp.puts           ; int puts(const char *s)
|       |   0x00400710      b800000000      mov eax, 0
|      ,==< 0x00400715      eb0f            jmp 0x400726
|      |`-> 0x00400717      bfe3074000      mov edi, str.Try_next_year_... ; 0x4007e3 ; "Try next year ..."
|      |    0x0040071c      e8fffdffff      call sym.imp.puts           ; int puts(const char *s)
|      |    0x00400721      b801000000      mov eax, 1
|      |      ; JMP XREF from 0x00400715 (main)
|      `--> 0x00400726      488b4df8        mov rcx, qword [local_8h]
|           0x0040072a      6448330c2528.   xor rcx, qword fs:[0x28]
|       ,=< 0x00400733      7405            je 0x40073a
|       |   0x00400735      e8f6fdffff      call sym.imp.__stack_chk_fail ; void __stack_chk_fail(void)
|       `-> 0x0040073a      c9              leave
\           0x0040073b      c3              ret
[0x7ff4442a3ed0]>
```

FIGURE 2. – pdfmain

Chose cool, il est possible de visualiser son code par blocs via `VV @main`.

2. Première résolution, simple analyse.

En regardant vite fait, on voit bien le processus du programme : 1. On demande a l'utilisateur un password

```
mov edi, str.Password_:     ; 0x4007c4 ; "Password : "
mov eax, 0
call sym.imp.printf         ; int printf(const char *format)
lea rax, [local_70h]
mov rsi, rax
mov edi, 0x4007d0
mov eax, 0
call sym.imp.__isoc99_scanf
```

via `printf` et `scanf`.

Puis on compare cette valeur avec une autre (`call sym.imp.strcmp ; int strcmp(const char *s1, const char *s2)`)

Et on félicite ou non la personne.

```
  ,=< 0x00400704      7511            jne 0x400717
  |   0x00400706      bfd3074000      mov edi, str.You_re_a_Wizard
      ; 0x4007d3 ; "You're a Wizard"
  |   0x0040070b      e810feffff      call sym.imp.puts           ;
      int puts(const char *s)
  |   0x00400710      b800000000      mov eax, 0
 ,==< 0x00400715      eb0f            jmp 0x400726
 |`-> 0x00400717      bfe3074000      mov edi, str.Try_next_year_...
     ; 0x4007e3 ; "Try next year ..."
 |    0x0040071c      e8fffdffff      call sym.imp.puts           ;
     int puts(const char *s)
 |    0x00400721      b801000000      mov eax, 1
```

On suppose simplement que la valeur qu'on cherche c'est le bon mot de passe et je m'intéresse donc à `strcmp`. En suivant les `mov` de dessus, on voit que `strcmp` compare le password qu'on à entré et une valeur que nous voyons en haut :

```
movabs rax, 0x3857397b47414c46
mov qword [local_80h], rax
mov dword [local_78h], 0x7a6e6273
mov word [local_74h], 0x7d38
mov byte [local_72h], 0
```

qui correspond a `FLAG{9W8sbnz8}` à l'envers, notre flag. Pour savoir pourquoi c'est à l'envers, je vous recommande les articles que Ge0  a pu faire sur ce site.

3. Seconde méthode, à l'exécution

Pour le coup, sans radare, j'aurais utilisé `gdb` en mettant un breakpoint et regardant le contenu de ce qui m'intéressait. Faisons donc ceci avec `r2`

Pour ceci, il faut lancer *radare* en mode debug (`-d` de tout à l'heure). Toutes les commandes de `r2` pour le debug commencent par `d`, voir ici : https ://radare.gitbooks.io/radare2book/content/introduction/basic_debugger_session.html . Ainsi prenons la ligne qui nous intéresse :

```
0x004006f2 e869feffff call sym.imp.strcmp ; int strcmp(const char *s1, const char *s2)
```

Puis, on peut lancer le programme avec `dc`. Une fois le breakpoint atteint, on peut alors récupérer le contenu de `rdx` contenant notre flag avec `px @ rdx` :

FIGURE 3. – px

4. Conclusion

Voilà pour ma petite découverte de radare, peut-etre la suite au prochain episode.