NAME : E . Blessing Charles

NO : 2019103012

Cryptography and Network Security

Finding Passwords in executables using GDB
[Architecture : x64]

1. Cracking simple password checker using strcmp
2. Cracking the hashed passwords

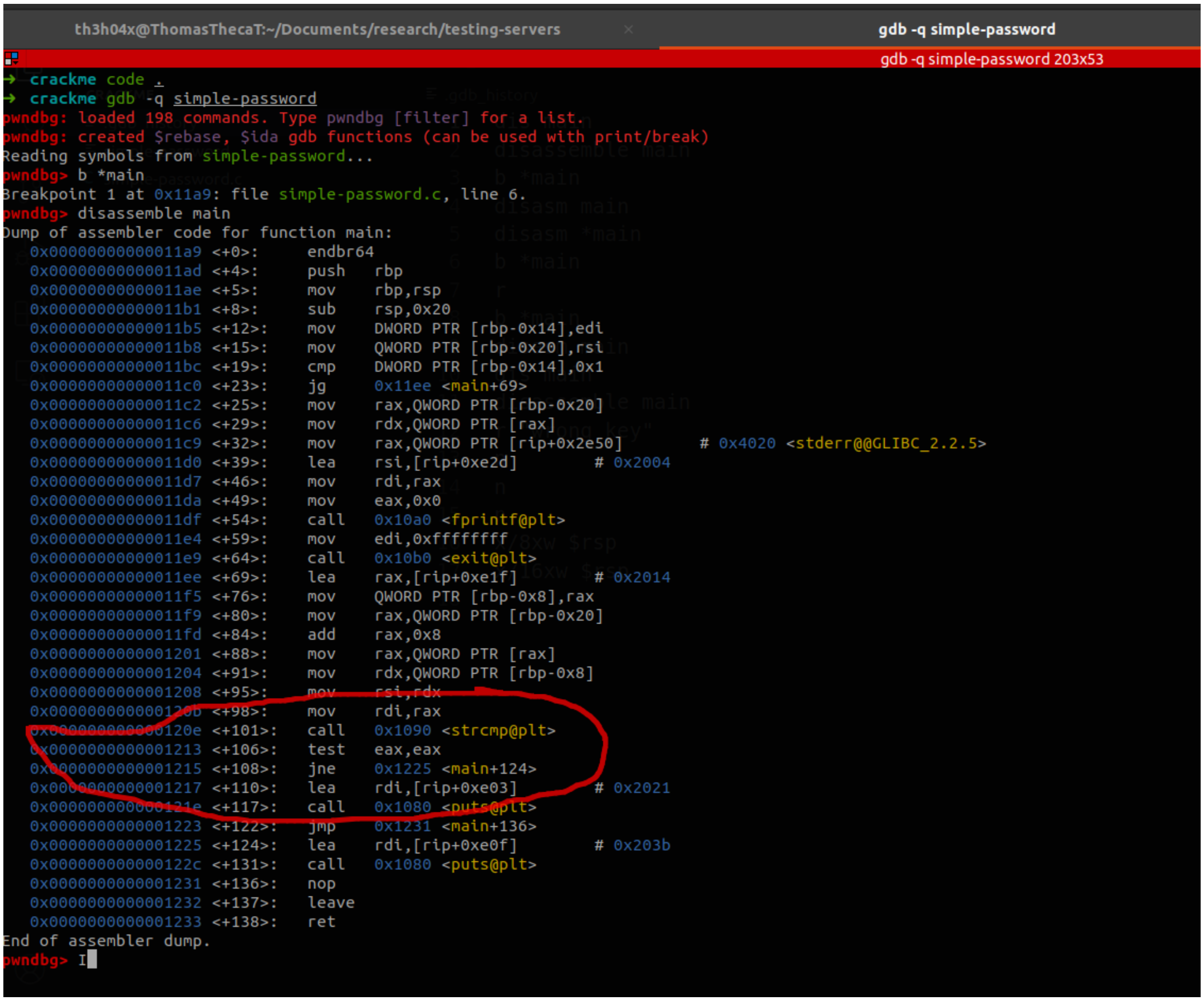
Cracking simple password checker using strcmp

C Code :

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>


void main(int argc , char **argv){
   if(argc < 2){
       fprintf(stderr , "%s <Password> \n" , argv[0]);
       exit(-1);
   }

   char * required_password = "thomasthecat" ;

   if(strcmp(argv[1] , required_password) == 0){
       printf("Hurrah , You cracked it !\n");
   }
   else
       printf("Aaaahh You failed : ( \n") ;

}
```

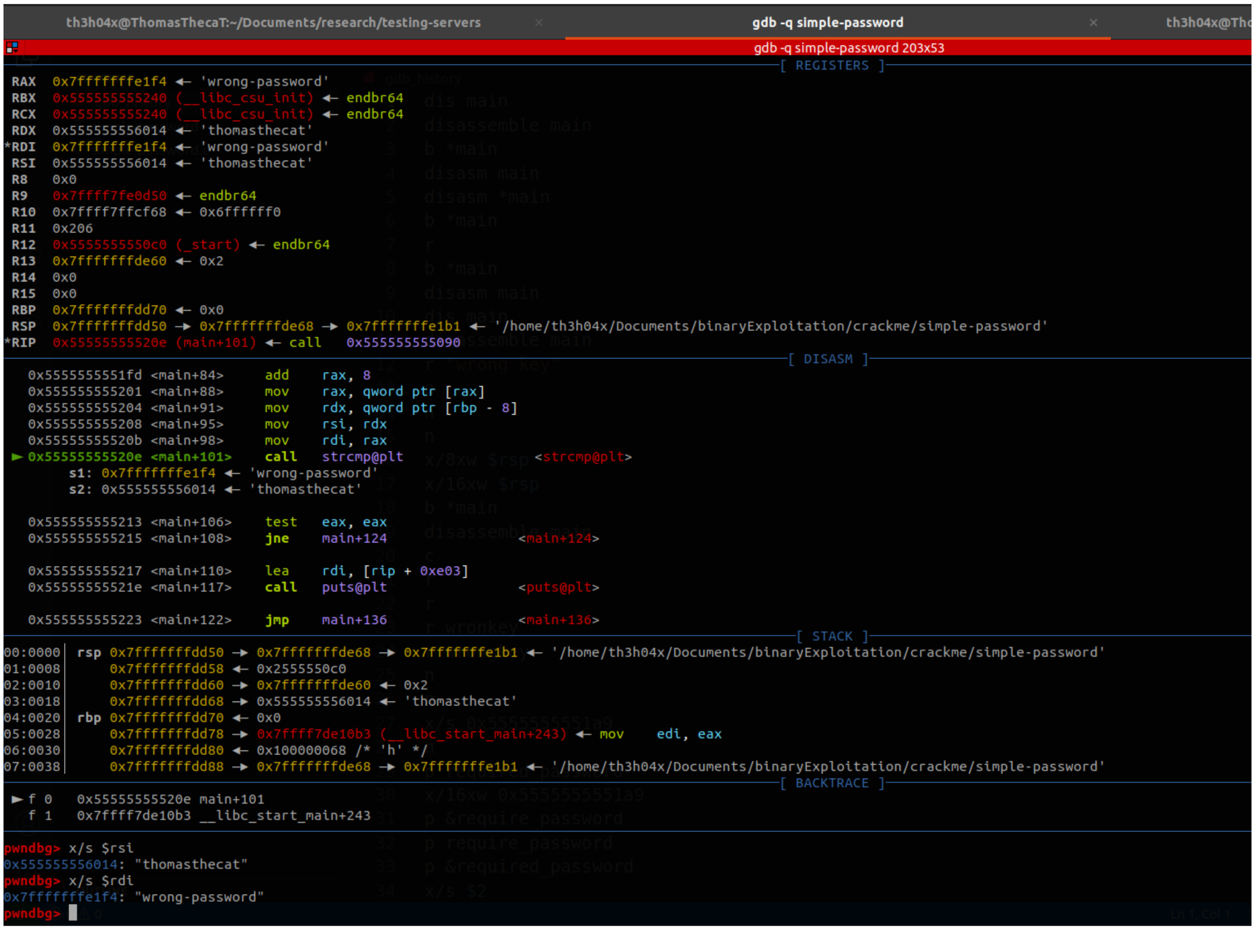
Cracking the password using GDB(pwndbg plugin):

Disassemble the main and we can see the strcmp is used to compare the password


th3h04x@ThomasThecaT:~/Documents/research/testing-servers
gdb -q simple-password
gdb -q simple-password 203x53

```
→ crackme code .
→ crackme gdb -q simple-password
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from simple-password...
pwndbg> b *main
Breakpoint 1 at 0x11a9: file simple-password.c, line 6.
pwndbg> disassemble main
Dump of assembler code for function main:
   0x00000000000011a9 <+0>:     endbr64
   0x00000000000011ad <+4>:     push   rbp
   0x00000000000011ae <+5>:     mov    rbp,rsp
   0x00000000000011b1 <+8>:     sub    rsp,0x20
   0x00000000000011b5 <+12>:    mov    DWORD PTR [rbp-0x14],edi
   0x00000000000011b8 <+15>:    mov    QWORD PTR [rbp-0x20],rsi
   0x00000000000011bc <+19>:    cmp    DWORD PTR [rbp-0x14],0x1
   0x00000000000011c0 <+23>:    jg     0x11ee <main+69>
   0x00000000000011c2 <+25>:    mov    rax,QWORD PTR [rbp-0x20]
   0x00000000000011c6 <+29>:    mov    rdx,QWORD PTR [rax]
   0x00000000000011c9 <+32>:    mov    rax,QWORD PTR [rip+0x2e50]        # 0x4020 <stderr@@GLIBC_2.2.5>
   0x00000000000011d0 <+39>:    lea    rsi,[rip+0xe2d]        # 0x2004
   0x00000000000011d7 <+46>:    mov    rdi,rax
   0x00000000000011da <+49>:    mov    eax,0x0
   0x00000000000011df <+54>:    call   0x10a0 <fprintf@plt>
   0x00000000000011e4 <+59>:    mov    edi,0xffffffff
   0x00000000000011e9 <+64>:    call   0x10b0 <exit@plt>
   0x00000000000011ee <+69>:    lea    rax,[rip+0xe1f]        # 0x2014
   0x00000000000011f5 <+76>:    mov    QWORD PTR [rbp-0x8],rax
   0x00000000000011f9 <+80>:    mov    rax,QWORD PTR [rbp-0x20]
   0x00000000000011fd <+84>:    add    rax,0x8
   0x0000000000001201 <+88>:    mov    rax,QWORD PTR [rax]
   0x0000000000001204 <+91>:    mov    rdx,QWORD PTR [rbp-0x8]
   0x0000000000001208 <+95>:    mov    rsi,rdx
   0x000000000000120b <+98>:    mov    rdi,rax
   0x000000000000120e <+101>:   call   0x1090 <strcmp@plt>
   0x0000000000001213 <+106>:   test   eax,eax
   0x0000000000001215 <+108>:   jne    0x1225 <main+124>
   0x0000000000001217 <+110>:   lea    rdi,[rip+0xe03]        # 0x2021
   0x000000000000121e <+117>:   call   0x1080 <puts@plt>
   0x0000000000001223 <+122>:   jmp    0x1231 <main+136>
   0x0000000000001225 <+124>:   lea    rdi,[rip+0xe0f]        # 0x203b
   0x000000000000122c <+131>:   call   0x1080 <puts@plt>
   0x0000000000001231 <+136>:   nop
   0x0000000000001232 <+137>:   leave
   0x0000000000001233 <+138>:   ret
End of assembler dump.
pwndbg> I
```

Before Calling the strcmp function the base address of the passed argument and the comparing password is moved into the rdi and rsi register . By examining the string we can see that the password is “thomasthecat”($rsi) and our custom input “wrong-password”($rdi) .



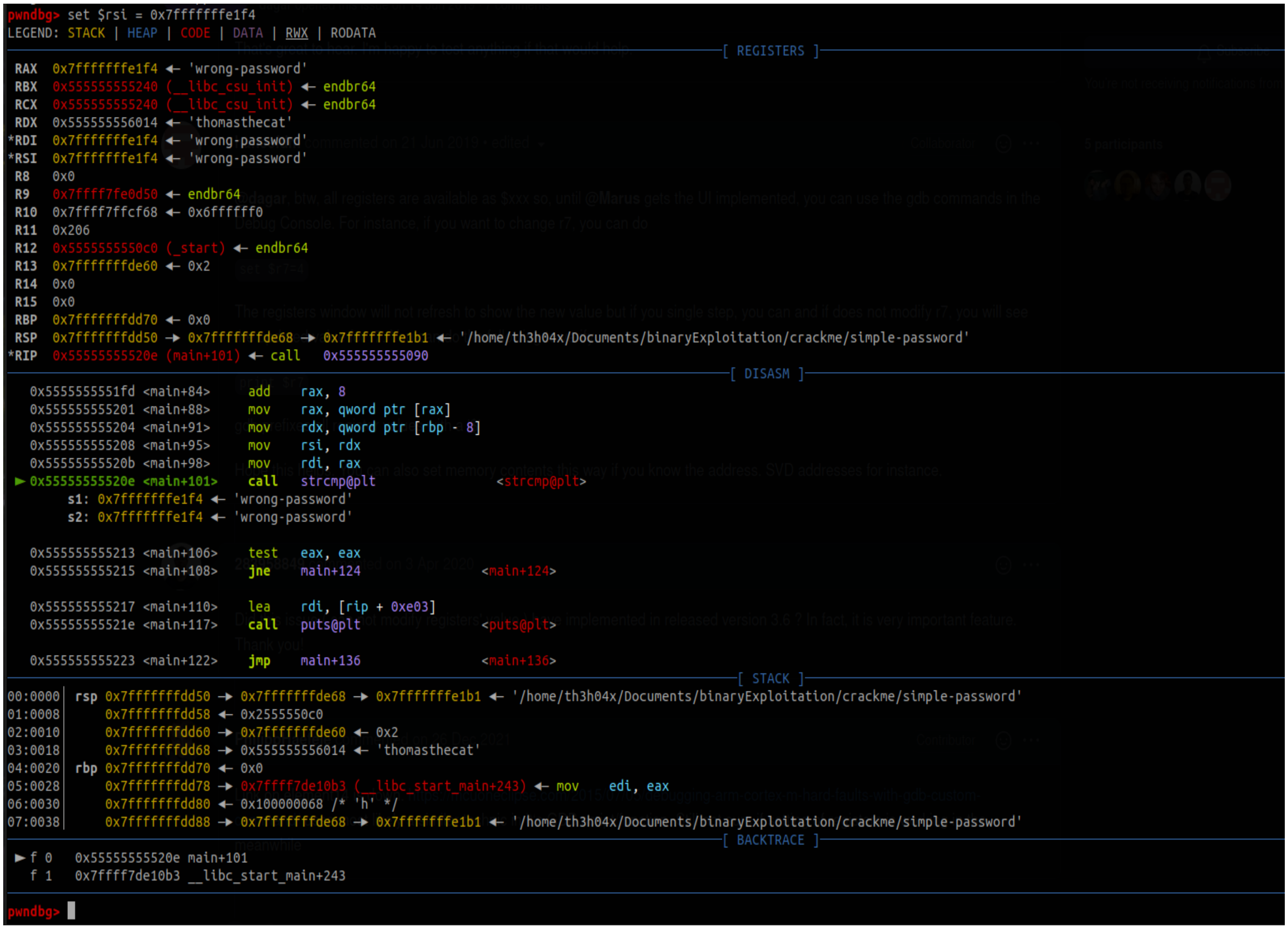
th3h04x@ThomasThecaT:~/Documents/research/testing-servers
gdb -q simple-password
th3h04x@Th
gdb -q simple-password 203x53

```
                                                                          [ REGISTERS ]
 RAX  0x7fffffffe1f4 ◂— 'wrong-password'
 RBX  0x555555555240 (__libc_csu_init) ◂— endbr64
 RCX  0x555555555240 (__libc_csu_init) ◂— endbr64
 RDX  0x555555556014 ◂— 'thomasthecat'
*RDI  0x7fffffffe1f4 ◂— 'wrong-password'
 RSI  0x555555556014 ◂— 'thomasthecat'
 R8   0x0
 R9   0x7ffff7fe0d50 ◂— endbr64
 R10  0x7ffff7ffcf68 ◂— 0x6ffffff0
 R11  0x206
 R12  0x5555555550c0 (_start) ◂— endbr64
 R13  0x7fffffffde60 ◂— 0x2
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdd70 ◂— 0x0
 RSP  0x7fffffffdd50 —▸ 0x7fffffffde68 —▸ 0x7fffffffe1b1 ◂— '/home/th3h04x/Documents/binaryExploitation/crackme/simple-password'
*RIP  0x55555555520e (main+101) ◂— call   0x555555555090
                                                                          [ DISASM ]
   0x5555555551fd <main+84>     add    rax, 8
   0x555555555201 <main+88>     mov    rax, qword ptr [rax]
   0x555555555204 <main+91>     mov    rdx, qword ptr [rbp - 8]
   0x555555555208 <main+95>     mov    rsi, rdx
   0x55555555520b <main+98>     mov    rdi, rax
 ► 0x55555555520e <main+101>    call   strcmp@plt                 <strcmp@plt>
        s1: 0x7fffffffe1f4 ◂— 'wrong-password'
        s2: 0x555555556014 ◂— 'thomasthecat'

   0x555555555213 <main+106>    test   eax, eax
   0x555555555215 <main+108>    jne    main+124                  <main+124>

   0x555555555217 <main+110>    lea    rdi, [rip + 0xe03]
   0x55555555521e <main+117>    call   puts@plt                  <puts@plt>

   0x555555555223 <main+122>    jmp    main+136                  <main+136>
                                                                          [ STACK ]
00:0000│ rsp 0x7fffffffdd50 —▸ 0x7fffffffde68 —▸ 0x7fffffffe1b1 ◂— '/home/th3h04x/Documents/binaryExploitation/crackme/simple-password'
01:0008│     0x7fffffffdd58 ◂— 0x2555550c0
02:0010│     0x7fffffffdd60 —▸ 0x7fffffffde60 ◂— 0x2
03:0018│     0x7fffffffdd68 —▸ 0x555555556014 ◂— 'thomasthecat'
04:0020│ rbp 0x7fffffffdd70 ◂— 0x0
05:0028│     0x7fffffffdd78 —▸ 0x7ffff7de10b3 (__libc_start_main+243) ◂— mov    edi, eax
06:0030│     0x7fffffffdd80 ◂— 0x100000068 /* 'h' */
07:0038│     0x7fffffffdd88 —▸ 0x7fffffffde68 —▸ 0x7fffffffe1b1 ◂— '/home/th3h04x/Documents/binaryExploitation/crackme/simple-password'
                                                                          [ BACKTRACE ]
 ► f 0   0x55555555520e main+101
   f 1   0x7ffff7de10b3 __libc_start_main+243

pwndbg> x/s $rsi
0x555555556014: "thomasthecat"
pwndbg> x/s $rdi
0x7fffffffe1f4: "wrong-password"
pwndbg> 
```

1. The user passed arg is at 0x7fffffffe1f4
2. The password is at 0x555555556014

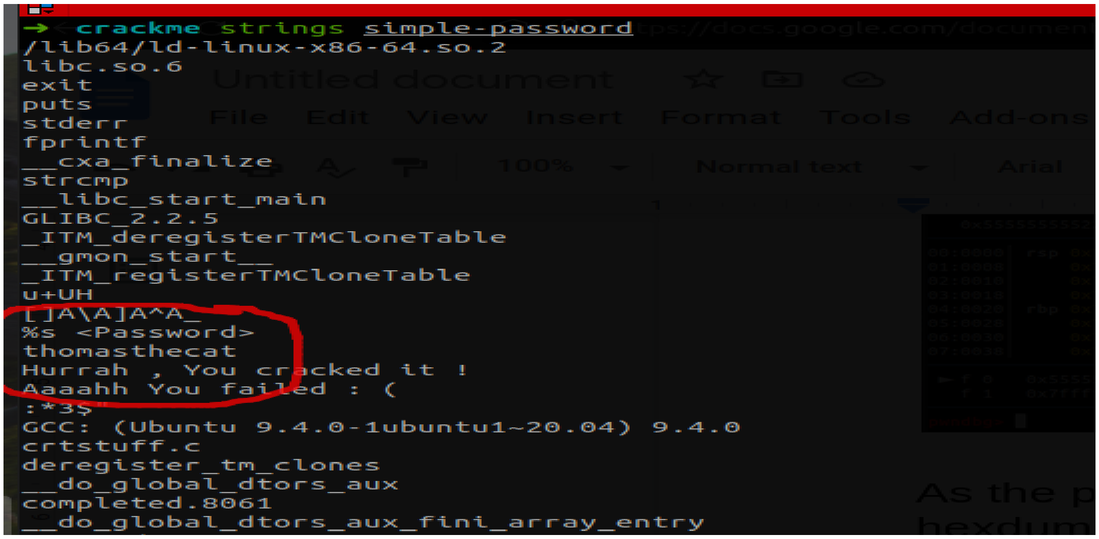
In gdb , we can directly set the rsi register which currently points to “thomas thecat” to our custom input by using the set command . Thus the strcmp check passes and we successfully cracked the executable .

The password is “thomasthecat” , which can also be passed directly via the cmdline args .



```
pwndbg> set $rsi = 0x7fffffffe1f4
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────────────────────[ REGISTERS ]───────────────────────────────────────────────
 RAX  0x7fffffffe1f4 ◂— 'wrong-password'
 RBX  0x555555555240 (__libc_csu_init) ◂— endbr64
 RCX  0x555555555240 (__libc_csu_init) ◂— endbr64
 RDX  0x555555556014 ◂— 'thomasthecat'
*RDI  0x7fffffffe1f4 ◂— 'wrong-password'
*RSI  0x7fffffffe1f4 ◂— 'wrong-password'
 R8   0x0
 R9   0x7ffff7fe0d50 ◂— endbr64
 R10  0x7ffff7ffcf68 ◂— 0x6ffffff0
 R11  0x206
 R12  0x5555555550c0 (_start) ◂— endbr64
 R13  0x7fffffffde60 ◂— 0x2
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdd70 ◂— 0x0
 RSP  0x7fffffffdd50 —▸ 0x7fffffffde68 —▸ 0x7fffffffe1b1 ◂— '/home/th3h04x/Documents/binaryExploitation/crackme/simple-password'
*RIP  0x55555555520e (main+101) ◂— call   0x555555555090
────────────────────────────────────────────────[ DISASM ]────────────────────────────────────────────────
   0x5555555551fd <main+84>     add    rax, 8
   0x555555555201 <main+88>     mov    rax, qword ptr [rax]
   0x555555555204 <main+91>     mov    rdx, qword ptr [rbp - 8]
   0x555555555208 <main+95>     mov    rsi, rdx
   0x55555555520b <main+98>     mov    rdi, rax
 ► 0x55555555520e <main+101>    call   strcmp@plt                <strcmp@plt>
        s1: 0x7fffffffe1f4 ◂— 'wrong-password'
        s2: 0x7fffffffe1f4 ◂— 'wrong-password'

   0x555555555213 <main+106>    test   eax, eax
   0x555555555215 <main+108>    jne    main+124                <main+124>

   0x555555555217 <main+110>    lea    rdi, [rip + 0xe03]
   0x55555555521e <main+117>    call   puts@plt                <puts@plt>

   0x555555555223 <main+122>    jmp    main+136                <main+136>
────────────────────────────────────────────────[ STACK ]─────────────────────────────────────────────────
00:0000│ rsp 0x7fffffffdd50 —▸ 0x7fffffffde68 —▸ 0x7fffffffe1b1 ◂— '/home/th3h04x/Documents/binaryExploitation/crackme/simple-password'
01:0008│     0x7fffffffdd58 ◂— 0x2555550c0
02:0010│     0x7fffffffdd60 —▸ 0x7fffffffde60 ◂— 0x2
03:0018│     0x7fffffffdd68 —▸ 0x555555556014 ◂— 'thomasthecat'
04:0020│ rbp 0x7fffffffdd70 ◂— 0x0
05:0028│     0x7fffffffdd78 —▸ 0x7ffff7de10b3 (__libc_start_main+243) ◂— mov    edi, eax
06:0030│     0x7fffffffdd80 ◂— 0x100000068 /* 'h' */
07:0038│     0x7fffffffdd88 —▸ 0x7fffffffde68 —▸ 0x7fffffffe1b1 ◂— '/home/th3h04x/Documents/binaryExploitation/crackme/simple-password'
──────────────────────────────────────────────[ BACKTRACE ]───────────────────────────────────────────────
 ► f 0   0x55555555520e main+101
   f 1   0x7ffff7de10b3 __libc_start_main+243
──────────────────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> 
```

As the password is hardcoded we can also directly use tools like strings and hexdump to see the strings directly .



```
→ crackme strings simple-password
/lib64/ld-linux-x86-64.so.2
libc.so.6
exit
puts
stderr
fprintf
__cxa_finalize
strcmp
__libc_start_main
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
u+UH
[]A\A]A^A_
%s <Password>
thomasthecat
Hurrah , You cracked it !
Aaaahh You failed : (
:*3$"
GCC: (Ubuntu 9.4.0-1ubuntu1~20.04) 9.4.0
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.8061
__do_global_dtors_aux_fini_array_entry
```

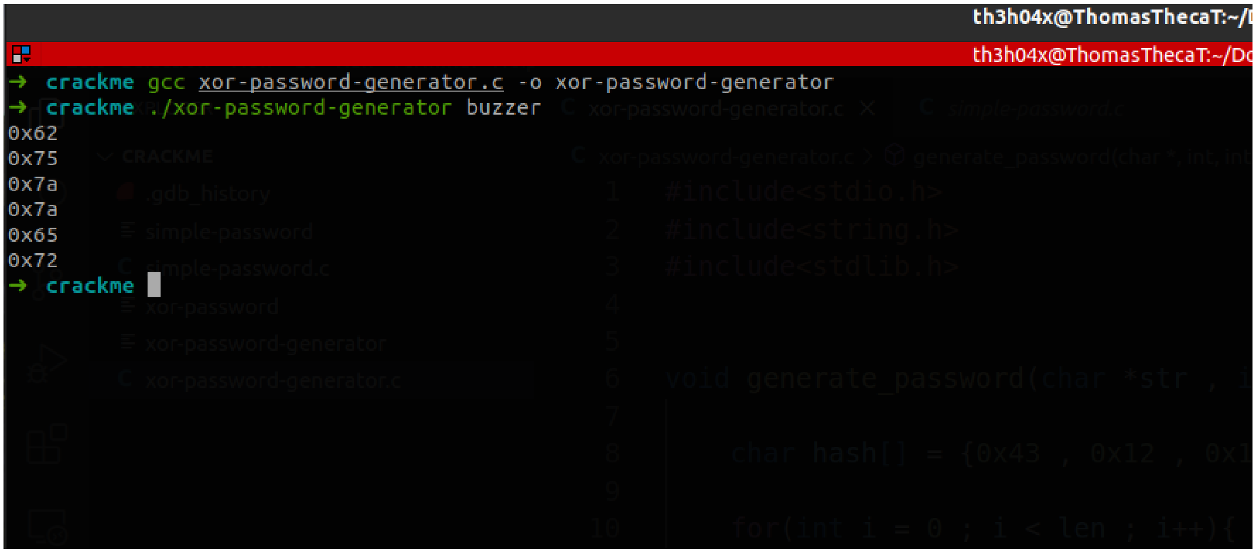
2. Cracking the hashed passwords

Instead of directly Hard coding the password in the executable , we can use some sort of algorithm to mangle the password . We use a simple hash array with random values which will be xor with the given key with n rotations to generate a new key .

C Code

```
void generate_password(char *str , int len , int
rotation_count){

   char hash[] = {0x43 , 0x12 , 0x17 , 0x42 , 0x18 , 0x12} ;

   for(int i = 0 ; i < len ; i++){
       for(int j = 0 ; j < rotation_count ; j++){
           str[i] = str[i] ^ hash[i] ;
       }
       printf("0x%x \n" , str[i]);
   }
}

void main(int argc , char **argv){
   if(argc < 2){
       fprintf(stderr , "%s <Password-to-encrypt> \n" ,
argv[0]);
       exit(-1);
   }
   generate_password(argv[1] , 6 , 2);

}
```

Generating the hashed password



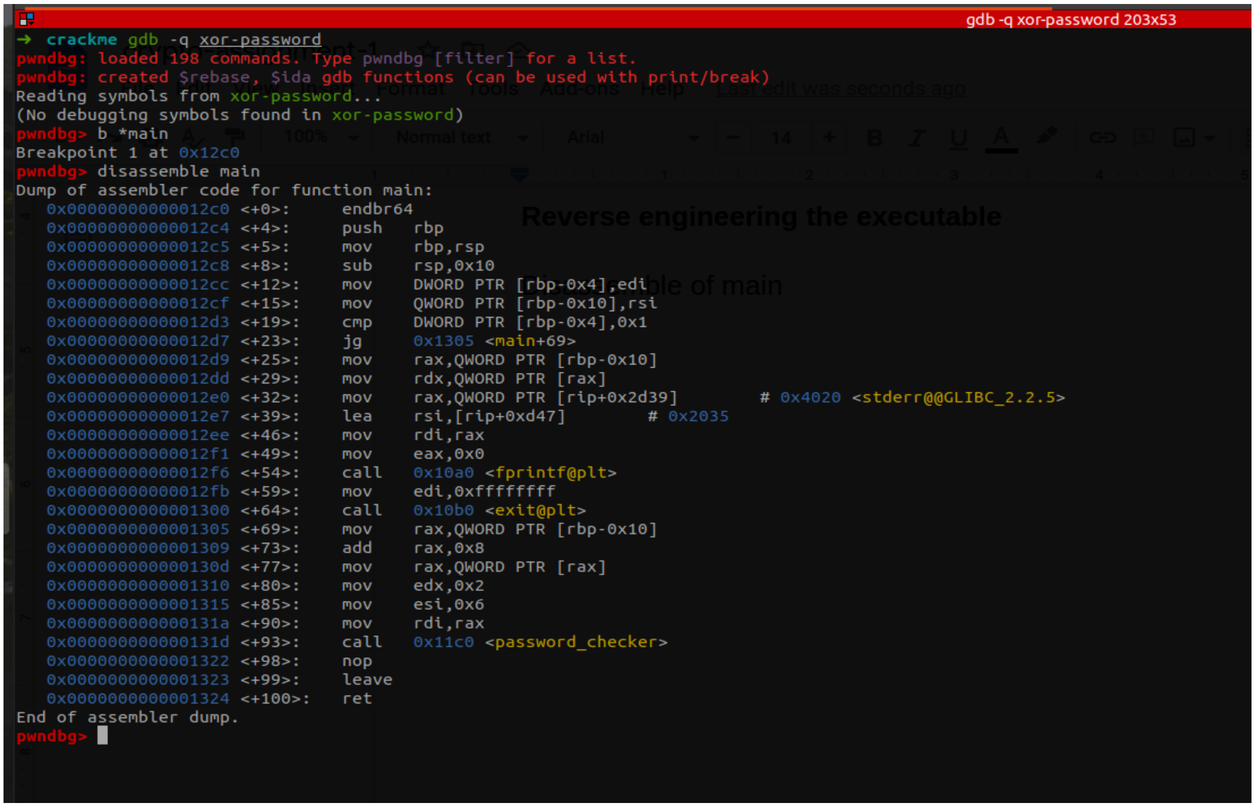
```
→ crackme gcc xor-password-generator.c -o xor-password-generator
→ crackme ./xor-password-generator buzzer
0x62
0x75
0x7a
0x7a
0x65
0x72
→ crackme 
```

Password Checker Code

```
void password_checker(char *str, int len, int rotation_count)
{
   char hash[] = {0x43, 0x12, 0x17, 0x42, 0x18, 0x12};
   char pass[] = {0x62, 0x75, 0x7a, 0x7a, 0x65, 0x72};
   for (int i = 0; i < len; i++)
   {
       for (int j = 0; j < rotation_count; j++)
       {
           str[i] = str[i] ^ hash[i];
       }
   }
   for (int i = 0; i < len; i++)
   {
       if (str[i] != pass[i])
       {
           printf("Password Checker Failed\n");
           exit(-1);
       }
   }
}
void main(int argc, char **argv)
{
   if (argc < 2)
   {
       fprintf(stderr, "%s <Password> \n", argv[0]);
       exit(-1);
   }
   succeed();
   password_checker(argv[1], 6, 2);
}
```

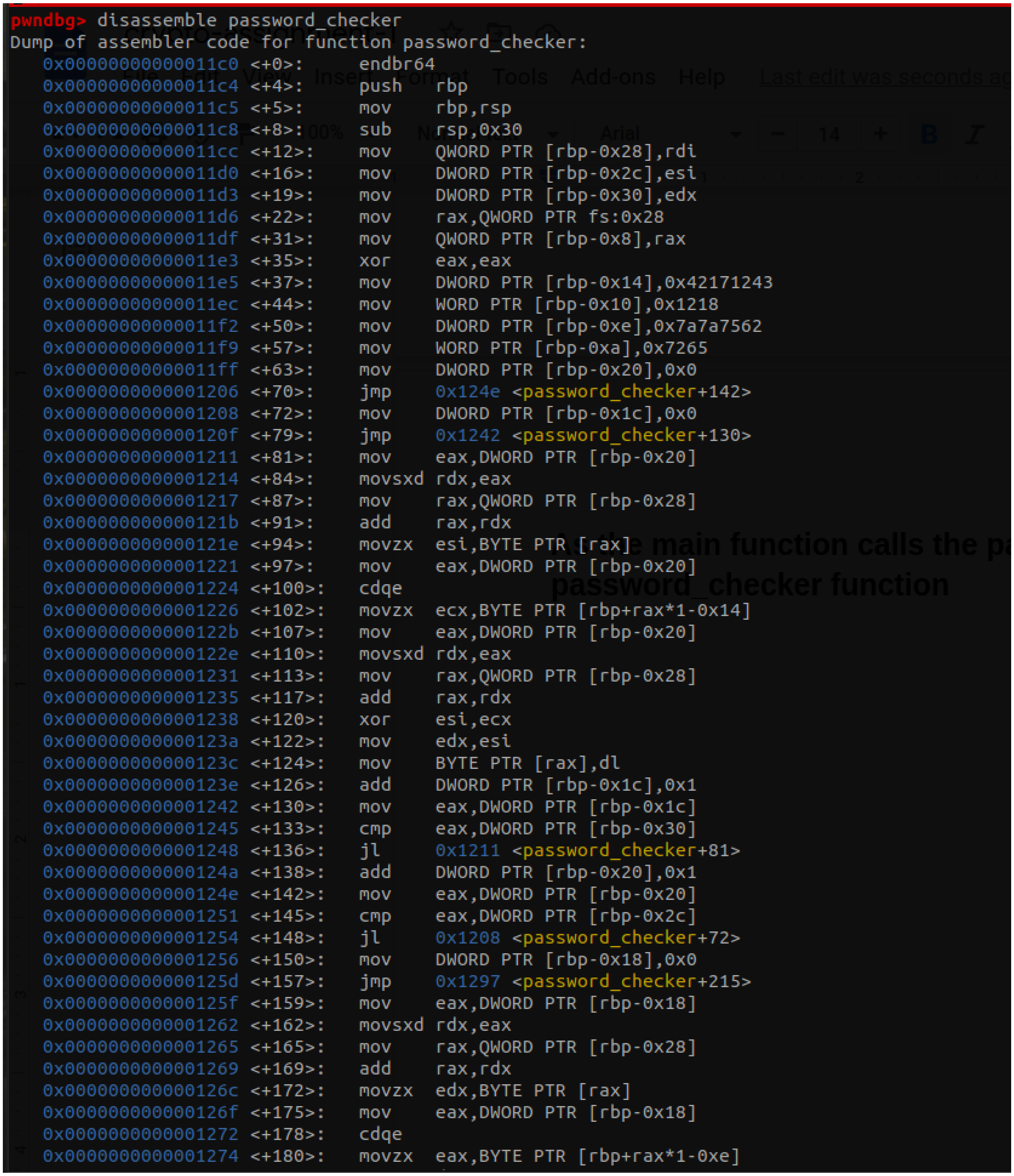
Reverse engineering the executable

Disassemble of main

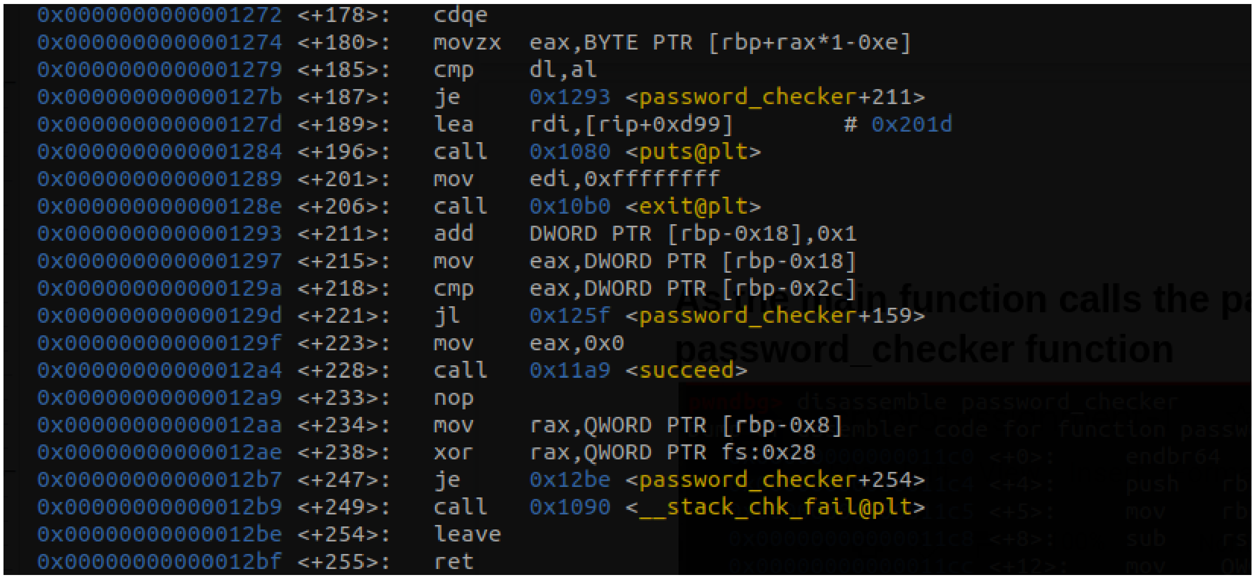


```
➜  crackme gdb -q xor-password
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from xor-password...
(No debugging symbols found in xor-password)
pwndbg> b *main
Breakpoint 1 at 0x12c0
pwndbg> disassemble main
Dump of assembler code for function main:
   0x00000000000012c0 <+0>:     endbr64
   0x00000000000012c4 <+4>:     push   rbp
   0x00000000000012c5 <+5>:     mov    rbp,rsp
   0x00000000000012c8 <+8>:     sub    rsp,0x10
   0x00000000000012cc <+12>:    mov    DWORD PTR [rbp-0x4],edi
   0x00000000000012cf <+15>:    mov    QWORD PTR [rbp-0x10],rsi
   0x00000000000012d3 <+19>:    cmp    DWORD PTR [rbp-0x4],0x1
   0x00000000000012d7 <+23>:    jg     0x1305 <main+69>
   0x00000000000012d9 <+25>:    mov    rax,QWORD PTR [rbp-0x10]
   0x00000000000012dd <+29>:    mov    rdx,QWORD PTR [rax]
   0x00000000000012e0 <+32>:    mov    rax,QWORD PTR [rip+0x2d39]        # 0x4020 <stderr@@GLIBC_2.2.5>
   0x00000000000012e7 <+39>:    lea    rsi,[rip+0xd47]        # 0x2035
   0x00000000000012ee <+46>:    mov    rdi,rax
   0x00000000000012f1 <+49>:    mov    eax,0x0
   0x00000000000012f6 <+54>:    call   0x10a0 <fprintf@plt>
   0x00000000000012fb <+59>:    mov    edi,0xffffffff
   0x0000000000001300 <+64>:    call   0x10b0 <exit@plt>
   0x0000000000001305 <+69>:    mov    rax,QWORD PTR [rbp-0x10]
   0x0000000000001309 <+73>:    add    rax,0x8
   0x000000000000130d <+77>:    mov    rax,QWORD PTR [rax]
   0x0000000000001310 <+80>:    mov    edx,0x2
   0x0000000000001315 <+85>:    mov    esi,0x6
   0x000000000000131a <+90>:    mov    rdi,rax
   0x000000000000131d <+93>:    call   0x11c0 <password_checker>
   0x0000000000001322 <+98>:    nop
   0x0000000000001323 <+99>:    leave
   0x0000000000001324 <+100>:   ret
End of assembler dump.
pwndbg> 
```

As the main function calls the password checker let disassemble the password_checker function



```
pwndbg> disassemble password_checker
Dump of assembler code for function password_checker:
   0x00000000000011c0 <+0>:     endbr64
   0x00000000000011c4 <+4>:     push   rbp
   0x00000000000011c5 <+5>:     mov    rbp,rsp
   0x00000000000011c8 <+8>:     sub    rsp,0x30
   0x00000000000011cc <+12>:    mov    QWORD PTR [rbp-0x28],rdi
   0x00000000000011d0 <+16>:    mov    DWORD PTR [rbp-0x2c],esi
   0x00000000000011d3 <+19>:    mov    DWORD PTR [rbp-0x30],edx
   0x00000000000011d6 <+22>:    mov    rax,QWORD PTR fs:0x28
   0x00000000000011df <+31>:    mov    QWORD PTR [rbp-0x8],rax
   0x00000000000011e3 <+35>:    xor    eax,eax
   0x00000000000011e5 <+37>:    mov    DWORD PTR [rbp-0x14],0x42171243
   0x00000000000011ec <+44>:    mov    WORD PTR [rbp-0x10],0x1218
   0x00000000000011f2 <+50>:    mov    DWORD PTR [rbp-0xe],0x7a7a7562
   0x00000000000011f9 <+57>:    mov    WORD PTR [rbp-0xa],0x7265
   0x00000000000011ff <+63>:    mov    DWORD PTR [rbp-0x20],0x0
   0x0000000000001206 <+70>:    jmp    0x124e <password_checker+142>
   0x0000000000001208 <+72>:    mov    DWORD PTR [rbp-0x1c],0x0
   0x000000000000120f <+79>:    jmp    0x1242 <password_checker+130>
   0x0000000000001211 <+81>:    mov    eax,DWORD PTR [rbp-0x20]
   0x0000000000001214 <+84>:    movsxd rdx,eax
   0x0000000000001217 <+87>:    mov    rax,QWORD PTR [rbp-0x28]
   0x000000000000121b <+91>:    add    rax,rdx
   0x000000000000121e <+94>:    movzx  esi,BYTE PTR [rax]
   0x0000000000001221 <+97>:    mov    eax,DWORD PTR [rbp-0x20]
   0x0000000000001224 <+100>:   cdqe
   0x0000000000001226 <+102>:   movzx  ecx,BYTE PTR [rbp+rax*1-0x14]
   0x000000000000122b <+107>:   mov    eax,DWORD PTR [rbp-0x20]
   0x000000000000122e <+110>:   movsxd rdx,eax
   0x0000000000001231 <+113>:   mov    rax,QWORD PTR [rbp-0x28]
   0x0000000000001235 <+117>:   add    rax,rdx
   0x0000000000001238 <+120>:   xor    esi,ecx
   0x000000000000123a <+122>:   mov    edx,esi
   0x000000000000123c <+124>:   mov    BYTE PTR [rax],dl
   0x000000000000123e <+126>:   add    DWORD PTR [rbp-0x1c],0x1
   0x0000000000001242 <+130>:   mov    eax,DWORD PTR [rbp-0x1c]
   0x0000000000001245 <+133>:   cmp    eax,DWORD PTR [rbp-0x30]
   0x0000000000001248 <+136>:   jl     0x1211 <password_checker+81>
   0x000000000000124a <+138>:   add    DWORD PTR [rbp-0x20],0x1
   0x000000000000124e <+142>:   mov    eax,DWORD PTR [rbp-0x20]
   0x0000000000001251 <+145>:   cmp    eax,DWORD PTR [rbp-0x2c]
   0x0000000000001254 <+148>:   jl     0x1208 <password_checker+72>
   0x0000000000001256 <+150>:   mov    DWORD PTR [rbp-0x18],0x0
   0x000000000000125d <+157>:   jmp    0x1297 <password_checker+215>
   0x000000000000125f <+159>:   mov    eax,DWORD PTR [rbp-0x18]
   0x0000000000001262 <+162>:   movsxd rdx,eax
   0x0000000000001265 <+165>:   mov    rax,QWORD PTR [rbp-0x28]
   0x0000000000001269 <+169>:   add    rax,rdx
   0x000000000000126c <+172>:   movzx  edx,BYTE PTR [rax]
   0x000000000000126f <+175>:   mov    eax,DWORD PTR [rbp-0x18]
   0x0000000000001272 <+178>:   cdqe
   0x0000000000001274 <+180>:   movzx  eax,BYTE PTR [rbp+rax*1-0xe]
```



```
   0x0000000000001272 <+178>:   cdqe
   0x0000000000001274 <+180>:   movzx  eax,BYTE PTR [rbp+rax*1-0xe]
   0x0000000000001279 <+185>:   cmp    dl,al
   0x000000000000127b <+187>:   je     0x1293 <password_checker+211>
   0x000000000000127d <+189>:   lea    rdi,[rip+0xd99]        # 0x201d
   0x0000000000001284 <+196>:   call   0x1080 <puts@plt>
   0x0000000000001289 <+201>:   mov    edi,0xffffffff
   0x000000000000128e <+206>:   call   0x10b0 <exit@plt>
   0x0000000000001293 <+211>:   add    DWORD PTR [rbp-0x18],0x1
   0x0000000000001297 <+215>:   mov    eax,DWORD PTR [rbp-0x18]
   0x000000000000129a <+218>:   cmp    eax,DWORD PTR [rbp-0x2c]
   0x000000000000129d <+221>:   jl     0x125f <password_checker+159>
   0x000000000000129f <+223>:   mov    eax,0x0
   0x00000000000012a4 <+228>:   call   0x11a9 <succeed>
   0x00000000000012a9 <+233>:   nop
   0x00000000000012aa <+234>:   mov    rax,QWORD PTR [rbp-0x8]
   0x00000000000012ae <+238>:   xor    rax,QWORD PTR fs:0x28
   0x00000000000012b7 <+247>:   je     0x12be <password_checker+254>
   0x00000000000012b9 <+249>:   call   0x1090 <__stack_chk_fail@plt>
   0x00000000000012be <+254>:   leave
   0x00000000000012bf <+255>:   ret
```

From the function password checker we can see , there is a function known as succeed , if we put the password correctly it will be called otherwise the exit function will be executed .

Important memory addresses to note

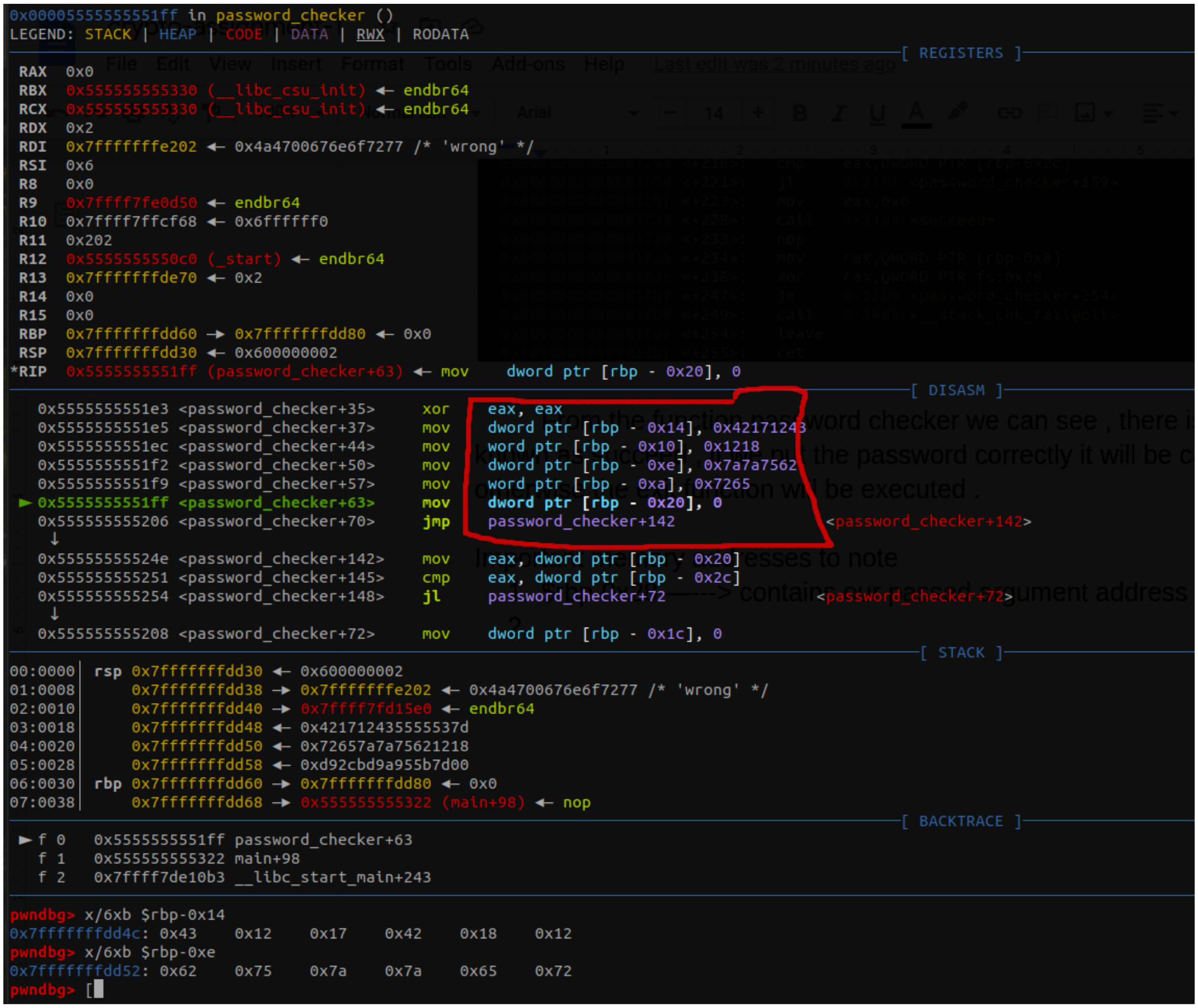
1. $rbp-0x28 ——--> contains our passed argument address

2. $rbp-0x14 ——---> The passed password is xored two times with the bytes in this location

3. $rbp-0xe ——----> The hashed password is then compared with the bytes in this location

   Thus our password must match after performing xor with $rbp-0x14 to $rbp-0xe

```
        Reverse engineered algo :
                while(2){
                        Hash_pwd = (Passed_arg  , $rbp-0x14)
                }
if(Hash_pwd == $rbp-0xe)   // success
```



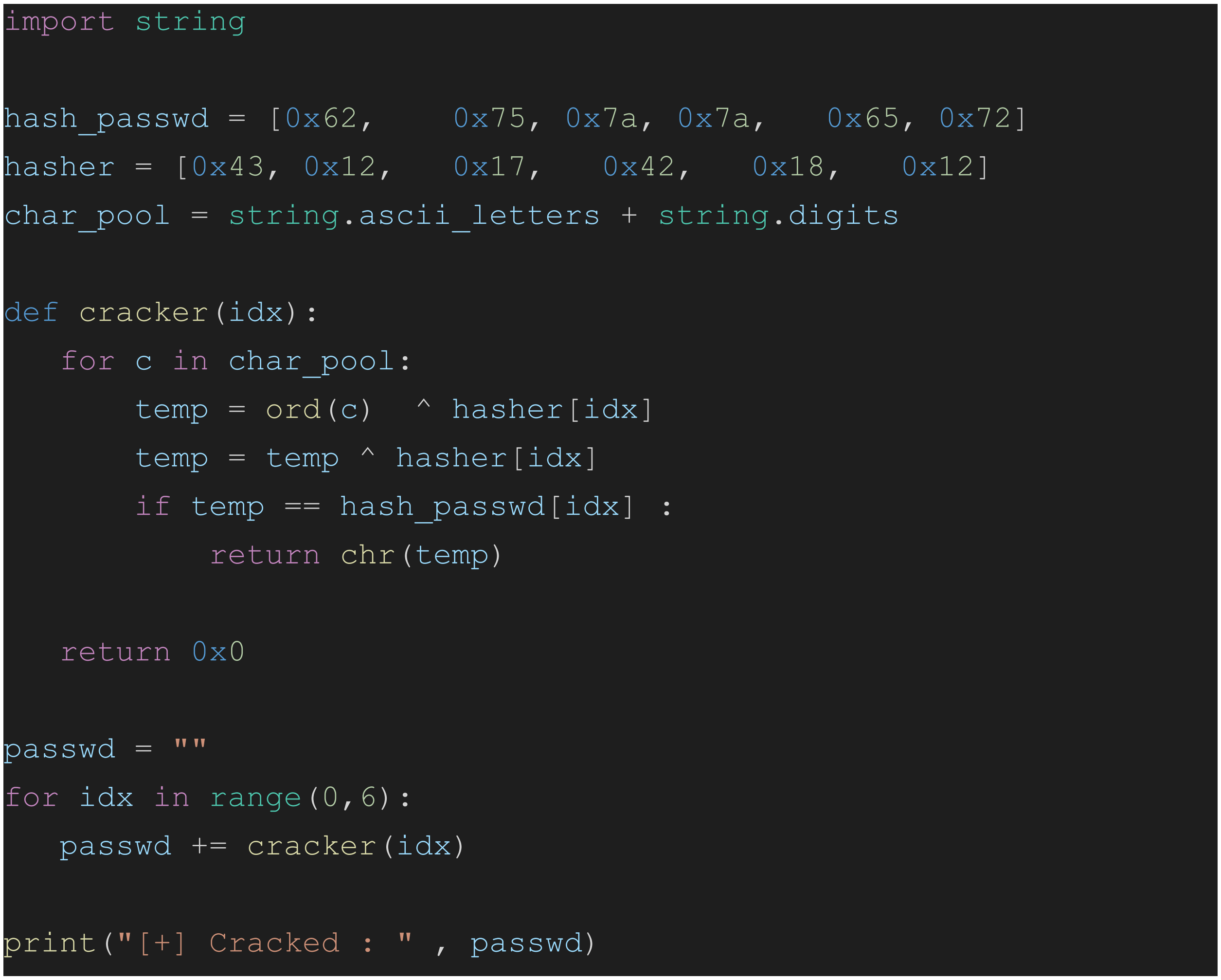
```
0x00005555555551ff in password_checker ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
──────────────────────────────────────────────────────[ REGISTERS ]──
 RAX  0x0
 RBX  0x555555555330 (__libc_csu_init) ◂— endbr64
 RCX  0x555555555330 (__libc_csu_init) ◂— endbr64
 RDX  0x2
 RDI  0x7fffffffe202 ◂— 0x4a4700676e6f7277 /* 'wrong' */
 RSI  0x6
 R8   0x0
 R9   0x7ffff7fe0d50 ◂— endbr64
 R10  0x7ffff7ffcf68 ◂— 0x6ffffff0
 R11  0x202
 R12  0x5555555550c0 (_start) ◂— endbr64
 R13  0x7fffffffde70 ◂— 0x2
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdd60 —▸ 0x7fffffffdd80 ◂— 0x0
 RSP  0x7fffffffdd30 ◂— 0x600000002
*RIP  0x5555555551ff (password_checker+63) ◂— mov    dword ptr [rbp - 0x20], 0
────────────────────────────────────────────────────────[ DISASM ]──
   0x5555555551e3 <password_checker+35>    xor    eax, eax
   0x5555555551e5 <password_checker+37>    mov    dword ptr [rbp - 0x14], 0x42171243
   0x5555555551ec <password_checker+44>    mov    word ptr [rbp - 0x10], 0x1218
   0x5555555551f2 <password_checker+50>    mov    dword ptr [rbp - 0xe], 0x7a7a7562
   0x5555555551f9 <password_checker+57>    mov    word ptr [rbp - 0xa], 0x7265
 ► 0x5555555551ff <password_checker+63>    mov    dword ptr [rbp - 0x20], 0
   0x555555555206 <password_checker+70>    jmp    password_checker+142                <password_checker+142>
    ↓
   0x55555555524e <password_checker+142>    mov    eax, dword ptr [rbp - 0x20]
   0x555555555251 <password_checker+145>    cmp    eax, dword ptr [rbp - 0x2c]
   0x555555555254 <password_checker+148>    jl     password_checker+72               <password_checker+72>
    ↓
   0x555555555208 <password_checker+72>    mov    dword ptr [rbp - 0x1c], 0
─────────────────────────────────────────────────────────[ STACK ]──
00:0000│ rsp 0x7fffffffdd30 ◂— 0x600000002
01:0008│     0x7fffffffdd38 —▸ 0x7fffffffe202 ◂— 0x4a4700676e6f7277 /* 'wrong' */
02:0010│     0x7fffffffdd40 —▸ 0x7ffff7fd15e0 ◂— endbr64
03:0018│     0x7fffffffdd48 ◂— 0x421712435555537d
04:0020│     0x7fffffffdd50 ◂— 0x72657a7a75621218
05:0028│     0x7fffffffdd58 ◂— 0xd92cbd9a955b7d00
06:0030│ rbp 0x7fffffffdd60 —▸ 0x7fffffffdd80 ◂— 0x0
07:0038│     0x7fffffffdd68 —▸ 0x555555555322 (main+98) ◂— nop
─────────────────────────────────────────────────────[ BACKTRACE ]──
 ► f 0   0x5555555551ff password_checker+63
   f 1   0x555555555322 main+98
   f 2   0x7ffff7de10b3 __libc_start_main+243
────────────────────────────────────────────────────────────────────
pwndbg> x/6xb $rbp-0x14
0x7fffffffdd4c: 0x43    0x12    0x17    0x42    0x18    0x12
pwndbg> x/6xb $rbp-0xe
0x7fffffffdd52: 0x62    0x75    0x7a    0x7a    0x65    0x72
pwndbg> [
```

From the above disassembly we can see bytes at the two memory locations

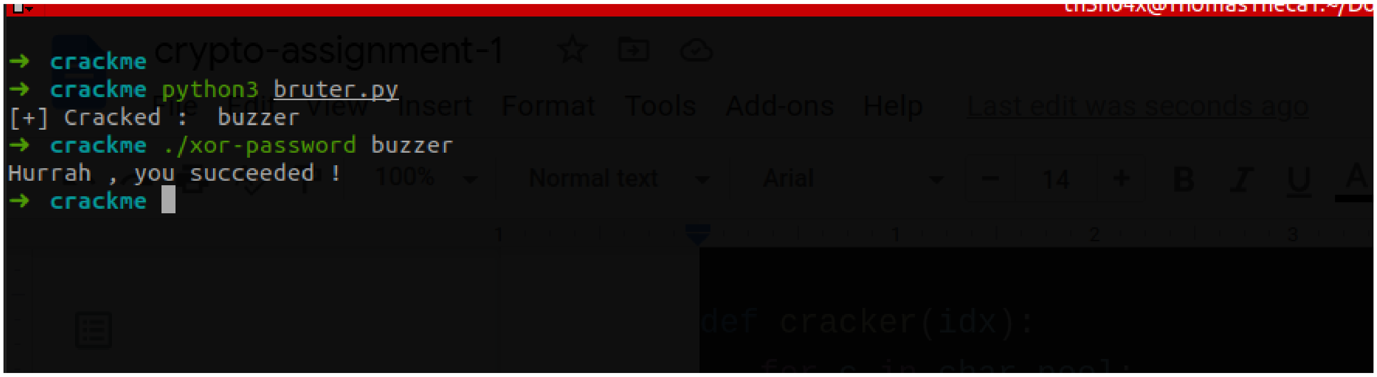
Hash   = {0x43   0x12   0x17   0x42   0x18   0x12}
Hashed_pwd = {0x62   0x75   0x7a   0x7a   0x65   0x72}

By writing a simple python bruteforcer we can crack the password

Python Script



```python
import string

hash_passwd = [0x62,    0x75, 0x7a, 0x7a,   0x65, 0x72]
hasher = [0x43, 0x12,   0x17,   0x42,   0x18,   0x12]
char_pool = string.ascii_letters + string.digits

def cracker(idx):
   for c in char_pool:
       temp = ord(c)  ^ hasher[idx]
       temp = temp ^ hasher[idx]
       if temp == hash_passwd[idx] :
           return chr(temp)

   return 0x0

passwd = ""
for idx in range(0,6):
   passwd += cracker(idx)

print("[+] Cracked : " , passwd)
```



th3h04x@ThomasTheCat:~/Do
crypto-assignment-1
Insert Format Tools Add-ons Help
Last edit was seconds ago
100%
Normal text
Arial
14

```
➜ crackme
➜ crackme python3 bruter.py
[+] Cracked :  buzzer
➜ crackme ./xor-password buzzer
Hurrah , you succeeded !
➜ crackme
```

The cracked password is buzzer