crackme03

1. This is from https://github.com/noracodes/crackmes. Special thanks to Noracode!
2. To create the executable,

```
make crackeme03
```

Writeup

Static analysis

Main

1. Here is the disassembly of main.

```
000000000000118d <main>:
    118d:       53                      push   rbx
    118e:       48 83 ec 10             sub    rsp,0x10
    1192:       83 ff 02                cmp    edi,0x2
    1195:       75 67                   jne    11fe <main+0x71>
    1197:       c7 44 24 09 6c 41 6d    mov    DWORD PTR [rsp+0x9],0x426d416c
    119e:       42
    119f:       c7 44 24 0c 42 64 41    mov    DWORD PTR [rsp+0xc],0x416442
    11a6:       00
    11a7:       c7 44 24 02 02 03 02    mov    DWORD PTR [rsp+0x2],0x3020302
    11ae:       03
    11af:       c7 44 24 05 03 05 00    mov    DWORD PTR [rsp+0x5],0x503
    11b6:       00
    11b7:       48 8b 5e 08             mov    rbx,QWORD PTR [rsi+0x8]
    11bb:       48 89 df                mov    rdi,rbx
    11be:       e8 7d fe ff ff          call   1040 <strlen@plt>
    11c3:       48 83 f8 06             cmp    rax,0x6
    11c7:       75 16                   jne    11df <main+0x52>
    11c9:       48 8d 54 24 02          lea    rdx,[rsp+0x2]
    11ce:       48 8d 74 24 09          lea    rsi,[rsp+0x9]
    11d3:       48 89 df                mov    rdi,rbx
    11d6:       e8 7e ff ff ff          call   1159 <check_pw>
    11db:       85 c0                   test   eax,eax
    11dd:       75 32                   jne    1211 <main+0x84>
    11df:       48 89 de                mov    rsi,rbx
    11e2:       48 8d 3d 4b 0e 00 00    lea    rdi,[rip+0xe4b]        # 2034
<_IO_stdin_used+0x34>
    11e9:       b8 00 00 00 00          mov    eax,0x0
    11ee:       e8 5d fe ff ff          call   1050 <printf@plt>
    11f3:       b8 01 00 00 00          mov    eax,0x1
    11f8:       48 83 c4 10             add    rsp,0x10
    11fc:       5b                      pop    rbx
    11fd:       c3                      ret
    11fe:       48 8d 3d ff 0d 00 00    lea    rdi,[rip+0xdff]        # 2004
<_IO_stdin_used+0x4>
    1205:       e8 26 fe ff ff          call   1030 <puts@plt>
    120a:       b8 ff ff ff ff          mov    eax,0xffffffff
    120f:       eb e7                   jmp    11f8 <main+0x6b>
    1211:       48 89 de                mov    rsi,rbx
    1214:       48 8d 3d 04 0e 00 00    lea    rdi,[rip+0xe04]        # 201f
<_IO_stdin_used+0x1f>
    121b:       b8 00 00 00 00          mov    eax,0x0
    1220:       e8 2b fe ff ff          call   1050 <printf@plt>
    1225:       b8 00 00 00 00          mov    eax,0x0
    122a:       eb cc                   jmp    11f8 <main+0x6b>
```

2. Let's start at the beginning.

```
    118d:       53                      push   rbx
    118e:       48 83 ec 10             sub    rsp,0x10
    1192:       83 ff 02                cmp    edi,0x2
    1195:       75 67                   jne    11fe <main+0x71>
```

- `sub rsp,0x10`: We increase the size of stack by 16 bytes
- `cmp edi,0x2`: `rdi` is the first argument when a function is called. In this case, it will contain the `argc` argument of `main`. So, here it is checking if `argc` is equal to 2. The name of the executable is the first argument by default, so really it is checking if 1 argument is passed in via the command line
- `jne 11fe <main+0x71>`: Jump here if `argc` is not 0.

3. Let's look at the assembly at `0x11fe`

```
    11fe:       48 8d 3d ff 0d 00 00    lea    rdi,[rip+0xdff]        # 2004
<_IO_stdin_used+0x4>
    1205:       e8 26 fe ff ff          call   1030 <puts@plt>
    120a:       b8 ff ff ff ff          mov    eax,0xffffffff
    120f:       eb e7                   jmp    11f8 <main+0x6b>
```

- Let's look what is the memory referenced by `rdi`

```
xxd -s 0x2004 -l 0x20 crackme03.64
00002004: 4e65 6564 2065 7861 6374 6c79 206f 6e65  Need exactly one
00002014: 2061 7267 756d 656e 742e 0059 6573 2c20   argument..Yes,
```

- `call 1030 <puts@plt>`: Here we call the puts function with the argument "Need exactly one argument"
- `mov eax,0xffffffff`: RAX contains -1

4. Let's look at the assembly at `11f8`

```
    11f8:       48 83 c4 10             add    rsp,0x10
    11fc:       5b                      pop    rbx
    11fd:       c3                      ret
```

- `add rsp,0x10`: Reduce the size of the stack

- We will return with `-1` because `eax` register contains `0xffffffff`
- In conclusion, it is something like this:

```
if(argc!=2){
        return -1;
}
```

5. Let's say we did not branch at `1195`

```
    1197:       c7 44 24 09 6c 41 6d    mov    DWORD PTR [rsp+0x9],0x426d416c
    119e:       42
    119f:       c7 44 24 0c 42 64 41    mov    DWORD PTR [rsp+0xc],0x416442
    11a6:       00
    11a7:       c7 44 24 02 02 03 02    mov    DWORD PTR [rsp+0x2],0x3020302
    11ae:       03
    11af:       c7 44 24 05 03 05 00    mov    DWORD PTR [rsp+0x5],0x503
    11b6:       00
    11b7:       48 8b 5e 08             mov    rbx,QWORD PTR [rsi+0x8]
    11bb:       48 89 df                mov    rdi,rbx
    11be:       e8 7d fe ff ff          call   1040 <strlen@plt>
    11c3:       48 83 f8 06             cmp    rax,0x6
    11c7:       75 16                   jne    11df <main+0x52>
```

- `[rsp+0x9]` will contain `BmAl`
- `[rsp+0xc]` will contain '\x00AdB'
- (After further investigation, I realised that there is an overlap of character, in this case B at `[rsp+0c]` and also it is little endian, so it is more like `lAmBdA\0`)
- `[rsp+0x2]` will contain the byte array [03,02,03,02,00,00,05,03]
- (After further investigation, I realised that there is an overlap of character, in this case 0x03 at `[rsp+05]` and also it is little endian, so it is more like `[02,03,02,03,03,05,00,00]`)
- `mov rbx,QWORD PTR [rsi+0x8]` : Contains 8 bytes starting from `[rsi+0x8]` . `rsi` is the second argument of `main` which is also `argv[0]` . `rsi+0x8` is `argv[1]`
- `call 1040 <strlen@plt>` : I believe we get the length of the string. The return value will be stored in the `rax` register
- `cmp rax,0x6` : Check if `rax` register contains value of 6
- `jne 11df <main+0x52>` : Jump if the value of `rax` is not 6.

6. Let's see what happens at `11df`

```
    11df:       48 89 de                mov    rsi,rbx
    11e2:       48 8d 3d 4b 0e 00 00    lea    rdi,[rip+0xe4b]        # 2034
<_IO_stdin_used+0x34>
    11e9:       b8 00 00 00 00          mov    eax,0x0
    11ee:       e8 5d fe ff ff          call   1050 <printf@plt>
    11f3:       b8 01 00 00 00          mov    eax,0x1
    11f8:       48 83 c4 10             add    rsp,0x10
    11fc:       5b                      pop    rbx
    11fd:       c3                      ret
```

- `rsi` will contain `argv[1]` . It will the second argument of `printf`

- `rdi` will contain the pointer to "No, %s is not correct"

```
xxd -s 0x2034 -l 0x20 crackme03.64
00002034: 4e6f 2c20 2573 2069 7320 6e6f 7420 636f  No, %s is not co
00002044: 7272 6563 742e 0a00 011b 033b 3000 0000  rrect......;0..
```

- `mov eax,0x0` : Since printf is a variadic function, `eax` will tell the function how many parameters is expected
- `call 1050 <printf@plt>` : Next, call printf
- Then, we return with 1 because `mov eax,0x1` (Return with error)
- In conclusion, it looks like this

```
if(strlen(argv[1])!=6){
        return 1;
}
```

7. Let's check what happens if length of string that we provided is 6. Let's see `11c9`

```
    11c9:       48 8d 54 24 02          lea    rdx,[rsp+0x2]
    11ce:       48 8d 74 24 09          lea    rsi,[rsp+0x9]
    11d3:       48 89 df                mov    rdi,rbx
    11d6:       e8 7e ff ff ff          call   1159 <check_pw>
    11db:       85 c0                   test   eax,eax
    11dd:       75 32                   jne    1211 <main+0x84>
```

- 3rd argument of a function call is stored in `rdx`. `rdx` will contain 8 bytes starting from `rsp+0x2`, that is [03,02,03,02,00,00,05,03]
- 2nd argument of a function call is stored in `rsi`. `rsi` will contain 8 bytes starting from `rsp+0x9`
- 1st argument of a function call is stored in `rdi`. `rdi` will contain `argv[1]`
- The output of `check_pw` is stored in `rax` register
- `test eax,eax` : Checks if `eax` is 0.
- `jne 1211 <main+0x84>` : Jump if it is not 0 to `0x1211`

8. Let's see what happens at `0x1211`

```
    1211:       48 89 de                mov    rsi,rbx
    1214:       48 8d 3d 04 0e 00 00    lea    rdi,[rip+0xe04]        # 201f
<_IO_stdin_used+0x1f>
    121b:       b8 00 00 00 00          mov    eax,0x0
    1220:       e8 2b fe ff ff          call   1050 <printf@plt>
    1225:       b8 00 00 00 00          mov    eax,0x0
    122a:       eb cc                   jmp    11f8 <main+0x6b>
```

- 1211, 1214, 121b is loading the arguments for printf at 1220
- `rdi` will contain the memory reference to

```
xxd -s 0x201f -l 0x20 crackme03.64
0000201f: 5965 732c 2025 7320 6973 2063 6f72 7265  Yes, %s is corre
0000202f: 6374 210a 004e 6f2c 2025 7320 6973 206e  ct!..No, %s is n
```

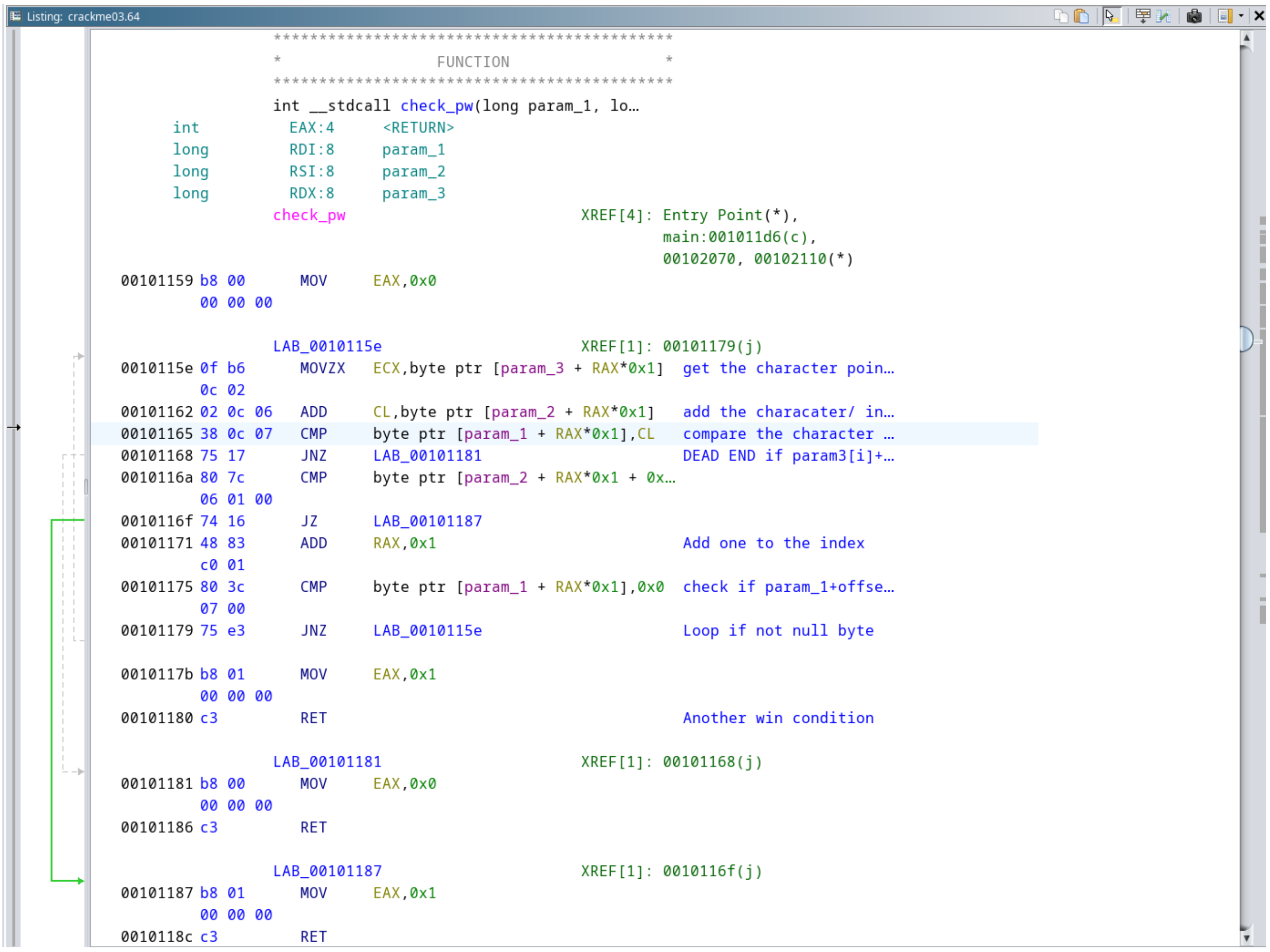
- `mov eax,0x0`: EAX is set to 0
- When moving to `11f8`, we will return with 0. (Successful return)
- So, we definitely want to come here.
- In conclusion,

```
if(argc!=2){
        return -1;
}
if(strlen(argv[1])!=6){
        return 1;
}
int[] intarray = [03,02,03,02,00,00,05,03]
if(check_pw(argv[1],"BmAl\0AdB",intarray)!=0){
        return 0;
}
```

check_pw

2. Here is another interesting function:

```
0000000000001159 <check_pw>:
    1159:       b8 00 00 00 00          mov    eax,0x0
    115e:       0f b6 0c 02             movzx  ecx,BYTE PTR [rdx+rax*1]
    1162:       02 0c 06                add    cl,BYTE PTR [rsi+rax*1]
    1165:       38 0c 07                cmp    BYTE PTR [rdi+rax*1],cl
    1168:       75 17                   jne    1181 <check_pw+0x28>
    116a:       80 7c 06 01 00          cmp    BYTE PTR [rsi+rax*1+0x1],0x0
    116f:       74 16                   je     1187 <check_pw+0x2e>
    1171:       48 83 c0 01             add    rax,0x1
    1175:       80 3c 07 00             cmp    BYTE PTR [rdi+rax*1],0x0
    1179:       75 e3                   jne    115e <check_pw+0x5>
    117b:       b8 01 00 00 00          mov    eax,0x1
    1180:       c3                      ret
    1181:       b8 00 00 00 00          mov    eax,0x0
    1186:       c3                      ret
    1187:       b8 01 00 00 00          mov    eax,0x1
    118c:       c3                      ret
```



```
                     **********************************************
                     *                  FUNCTION                  *
                     **********************************************
                     int __stdcall check_pw(long param_1, lo…
        int           EAX:4      <RETURN>
        long          RDI:8      param_1
        long          RSI:8      param_2
        long          RDX:8      param_3
                     check_pw                         XREF[4]: Entry Point(*),
                                                                main:001011d6(c),
                                                                00102070, 00102110(*)
00101159 b8 00       MOV     EAX,0x0
         00 00 00

                     LAB_0010115e                     XREF[1]: 00101179(j)
0010115e 0f b6       MOVZX   ECX,byte ptr [param_3 + RAX*0x1]  get the character poin…
         0c 02
00101162 02 0c 06    ADD     CL,byte ptr [param_2 + RAX*0x1]   add the characater/ in…
00101165 38 0c 07    CMP     byte ptr [param_1 + RAX*0x1],CL   compare the character …
00101168 75 17       JNZ     LAB_00101181                       DEAD END if param3[i]+…
0010116a 80 7c       CMP     byte ptr [param_2 + RAX*0x1 + 0x…
         06 01 00
0010116f 74 16       JZ      LAB_00101187
00101171 48 83       ADD     RAX,0x1                            Add one to the index
         c0 01
00101175 80 3c       CMP     byte ptr [param_1 + RAX*0x1],0x0  check if param_1+offse…
         07 00
00101179 75 e3       JNZ     LAB_0010115e                       Loop if not null byte

0010117b b8 01       MOV     EAX,0x1
         00 00 00
00101180 c3          RET                                        Another win condition

                     LAB_00101181                     XREF[1]: 00101168(j)
00101181 b8 00       MOV     EAX,0x0
         00 00 00
00101186 c3          RET

                     LAB_00101187                     XREF[1]: 0010116f(j)
00101187 b8 01       MOV     EAX,0x1
         00 00 00
0010118c c3          RET
```

Solution

1. Let's try to solve the challenge by adding ciphertext[i] with mask[i]

```
ciphertext=bytearray.fromhex('00416442426d416c')
mask=bytearray.fromhex('0302030200000503')

for i in range(len(ciphertext)):
        print(bytearray.fromhex("{:02x}".format(ciphertext[i]+mask[i])).decode(), end='')
```

Output:

```
CgDBmFo
```

2. Let's send this to the binary.

```
./crackme03.64 CgDBmFo
```

Output:

```
No, CgDBmFo is not correct.
```

3. Alright, I think it is little endian

```python
ciphertext=bytearray.fromhex('6c416d4242644100')
mask=bytearray.fromhex('0203020303050000')

for i in range(len(ciphertext)):
        print(bytearray.fromhex("{:02x}".format(ciphertext[i]+mask[i])).decode(), end='')
```

- It produced `nDoEEiA`
  Output:

```
 ./crackme03.64 nDoEEiA
No, nDoEEiA is not correct.
```