ATTACK DEFENSE

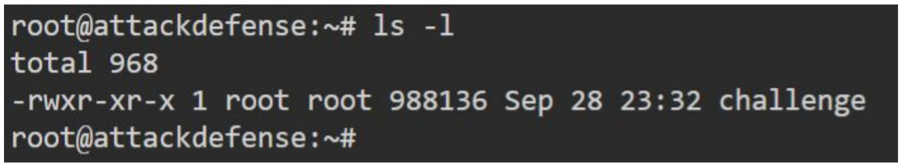
by PentesterAcademy

Name	Caesar Protection
URL	https://www.attackdefense.com/challengedetails?cid=107
Type	Reserve Engineering : Static Binary Analysis

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic.

Step 1: Check the given file.

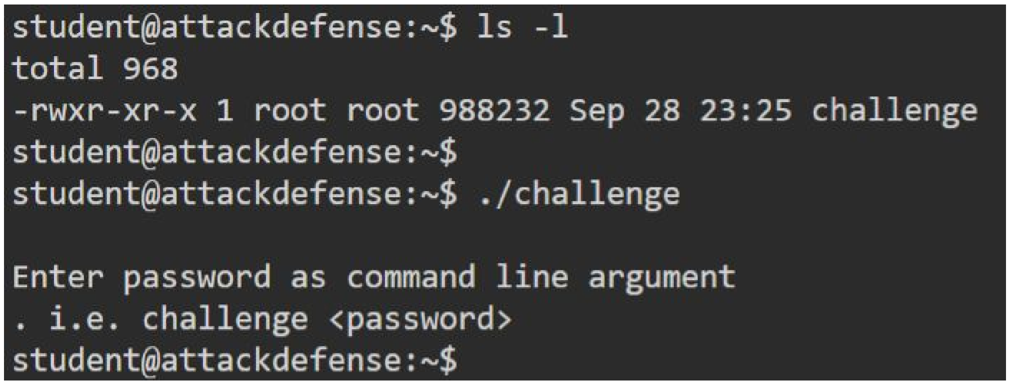
Command: ls -l



```
root@attackdefense:~# ls -l
total 968
-rwxr-xr-x 1 root root 988136 Sep 28 23:32 challenge
root@attackdefense:~#
```

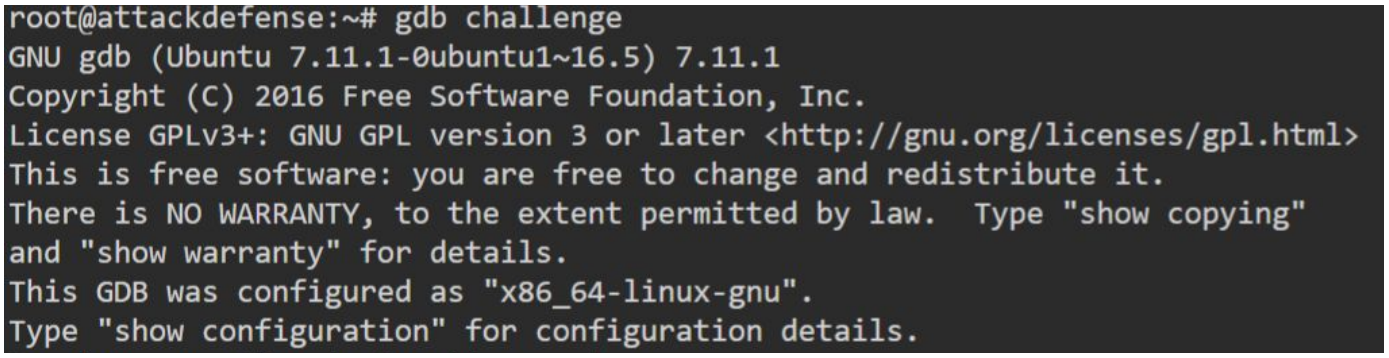
Step 2: Execute it. One needs to pass the correct passcode to the binary in order to get the information.

Command: ./challenge

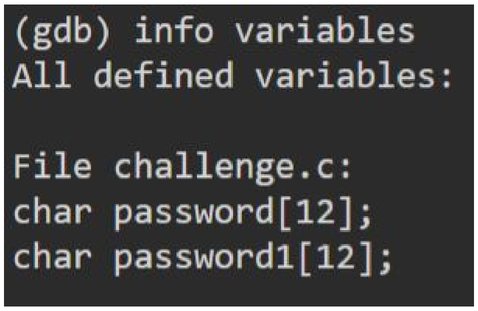


```
student@attackdefense:~$ ls -l
total 968
-rwxr-xr-x 1 root root 988232 Sep 28 23:25 challenge
student@attackdefense:~$
student@attackdefense:~$ ./challenge

Enter password as command line argument
. i.e. challenge <password>
student@attackdefense:~$
```

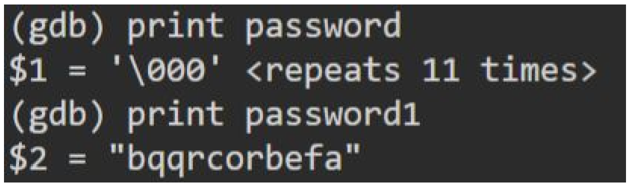
Step 3: Open this binary in GDB.


```
root@attackdefense:~# gdb challenge
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
```

Print the names of the global variables used in the binary.

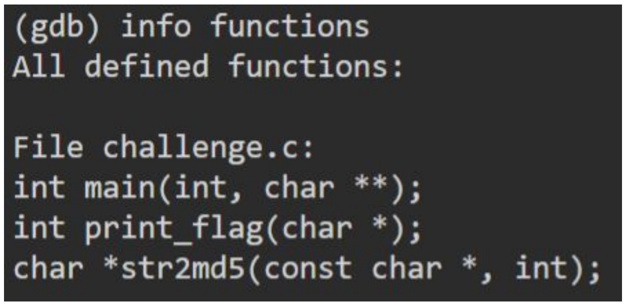

```
(gdb) info variables
All defined variables:

File challenge.c:
char password[12];
char password1[12];
```

Print variables


```
(gdb) print password
$1 = '\000' <repeats 11 times>
(gdb) print password1
$2 = "bqqrcorbefa"
```

Print the list of functions

Command: info functions



```
(gdb) info functions
All defined functions:

File challenge.c:
int main(int, char **);
int print_flag(char *);
char *str2md5(const char *, int);
```

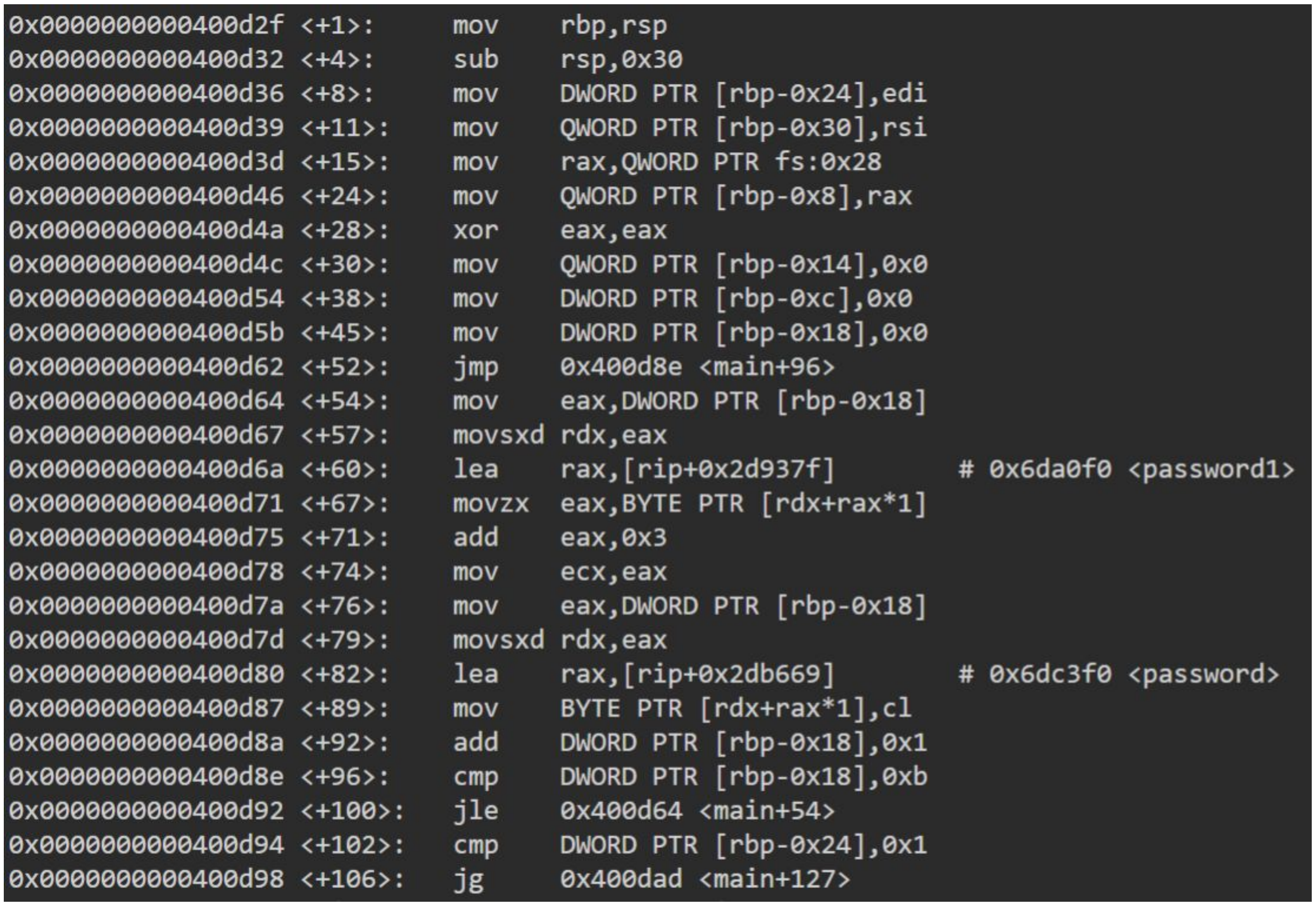
Step 4: Set disassembly flavor to Intel style (It is more user friendly and known then default ATT style).

Command: set disassembly-flavor intel

Disassemble main() function

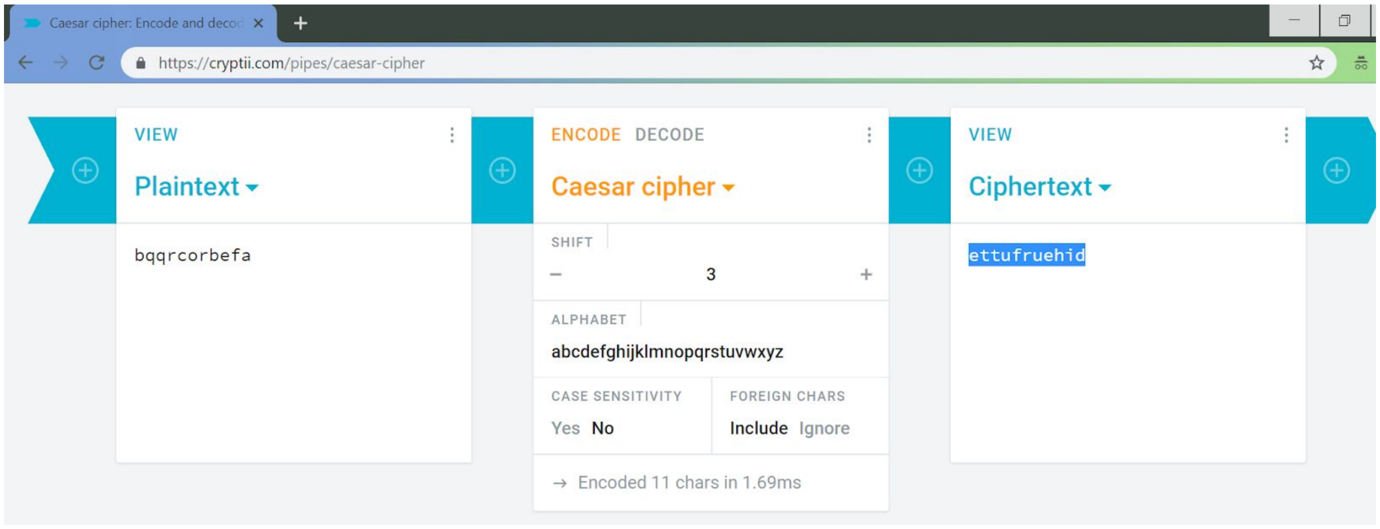
Command: disassemble main

By looking closely on the disassembled code, it is clear that the characters are being incremented by 3 to create the final password.



```
0x0000000000400d2f <+1>:     mov    rbp,rsp
0x0000000000400d32 <+4>:     sub    rsp,0x30
0x0000000000400d36 <+8>:     mov    DWORD PTR [rbp-0x24],edi
0x0000000000400d39 <+11>:    mov    QWORD PTR [rbp-0x30],rsi
0x0000000000400d3d <+15>:    mov    rax,QWORD PTR fs:0x28
0x0000000000400d46 <+24>:    mov    QWORD PTR [rbp-0x8],rax
0x0000000000400d4a <+28>:    xor    eax,eax
0x0000000000400d4c <+30>:    mov    QWORD PTR [rbp-0x14],0x0
0x0000000000400d54 <+38>:    mov    DWORD PTR [rbp-0xc],0x0
0x0000000000400d5b <+45>:    mov    DWORD PTR [rbp-0x18],0x0
0x0000000000400d62 <+52>:    jmp    0x400d8e <main+96>
0x0000000000400d64 <+54>:    mov    eax,DWORD PTR [rbp-0x18]
0x0000000000400d67 <+57>:    movsxd rdx,eax
0x0000000000400d6a <+60>:    lea    rax,[rip+0x2d937f]        # 0x6da0f0 <password1>
0x0000000000400d71 <+67>:    movzx  eax,BYTE PTR [rdx+rax*1]
0x0000000000400d75 <+71>:    add    eax,0x3
0x0000000000400d78 <+74>:    mov    ecx,eax
0x0000000000400d7a <+76>:    mov    eax,DWORD PTR [rbp-0x18]
0x0000000000400d7d <+79>:    movsxd rdx,eax
0x0000000000400d80 <+82>:    lea    rax,[rip+0x2db669]        # 0x6dc3f0 <password>
0x0000000000400d87 <+89>:    mov    BYTE PTR [rdx+rax*1],cl
0x0000000000400d8a <+92>:    add    DWORD PTR [rbp-0x18],0x1
0x0000000000400d8e <+96>:    cmp    DWORD PTR [rbp-0x18],0xb
0x0000000000400d92 <+100>:   jle    0x400d64 <main+54>
0x0000000000400d94 <+102>:   cmp    DWORD PTR [rbp-0x24],0x1
0x0000000000400d98 <+106>:   jg     0x400dad <main+127>
```

Step 5: Convert the string in password1 (bqqrcorbefa) to caesar cipher with shift of 3. This will give us ettufruehid as passcode.



Caesar cipher: Encode and decod
https://cryptii.com/pipes/caesar-cipher
VIEW
Plaintext
ENCODE DECODE
Caesar cipher
VIEW
Ciphertext

```
bqqrcorbefa
```

SHIFT 3

ALPHABET abcdefghijklmnopqrstuvwxyz

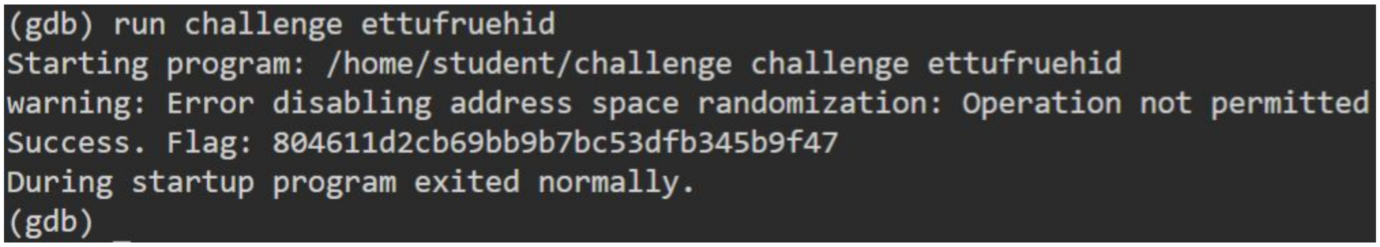
CASE SENSITIVITY Yes No

FOREIGN CHARS Include Ignore

→ Encoded 11 chars in 1.69ms

```
ettufruehid
```

Step 6: Trying this passphrase



```
(gdb) run challenge ettufruehid
Starting program: /home/student/challenge challenge ettufruehid
warning: Error disabling address space randomization: Operation not permitted
Success. Flag: 804611d2cb69bb9b7bc53dfb345b9f47
During startup program exited normally.
(gdb) _
```

Flag: 804611d2cb69bb9b7bc53dfb345b9f47

References:

1. GDB (https://www.gnu.org/software/gdb/)