
ATTACK
DEFENSE
by PentesterAcademy

Name	Garbled Password
URL	https://www.attackdefense.com/challengedetails?cid=106
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

```
root@attackdefense:~# ./challenge

Enter password as command line argument

i.e. challenge <password>
root@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.
```

Step 4: Print the names of the global variables used in the binary.

Command: info variables

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

File challenge.c:
char password[13];
char password1[12];
char password2[12];
```

Print these Variables.

Command: print password

```
(gdb) print password
$1 = '\000' <repeats 12 times>
(gdb) print password1
$2 = "n1v2r3i4e5p6"
(gdb) print password2
$3 = "1e2e3g4v5u6!"
```

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);
int validate_password(char *);
```

Step 5: Set disassembly flavor to Intel style (It is more user friendly and known then default ATT style).

Command: set disassembly-flavor intel

Disassemble validate_password() function

Command: disassemble validate_password

Step 6: By looking closely on the disassembled code, it is clear that characters are being moved from password1 and password2 to password one by one. So, the final password is made up by mixing both password1 and password2.

```
0x0000000000400d36 <+8>:     mov    QWORD PTR [rbp-0x18],rdi
0x0000000000400d3a <+12>:    mov    DWORD PTR [rbp-0x4],0x0
0x0000000000400d41 <+19>:    jmp    0x400d95 <validate_password+103>
0x0000000000400d43 <+21>:    mov    eax,DWORD PTR [rbp-0x4]
0x0000000000400d46 <+24>:    and    eax,0x1
0x0000000000400d49 <+27>:    test   eax,eax
0x0000000000400d4b <+29>:    je     0x400d70 <validate_password+66>
0x0000000000400d4d <+31>:    mov    eax,DWORD PTR [rbp-0x4]
0x0000000000400d50 <+34>:    movsxd rdx,eax
0x0000000000400d53 <+37>:    lea    rax,[rip+0x2d93a6]        # 0x6da100 <password2>
0x0000000000400d5a <+44>:    movzx  ecx,BYTE PTR [rdx+rax*1]
0x0000000000400d5e <+48>:    mov    eax,DWORD PTR [rbp-0x4]
0x0000000000400d61 <+51>:    movsxd rdx,eax
0x0000000000400d64 <+54>:    lea    rax,[rip+0x2db6a5]        # 0x6dc410 <password>
0x0000000000400d6b <+61>:    mov    BYTE PTR [rdx+rax*1],cl
0x0000000000400d6e <+64>:    jmp    0x400d91 <validate_password+99>
0x0000000000400d70 <+66>:    mov    eax,DWORD PTR [rbp-0x4]
0x0000000000400d73 <+69>:    movsxd rdx,eax
0x0000000000400d76 <+72>:    lea    rax,[rip+0x2d9373]        # 0x6da0f0 <password1>
0x0000000000400d7d <+79>:    movzx  ecx,BYTE PTR [rdx+rax*1]
0x0000000000400d81 <+83>:    mov    eax,DWORD PTR [rbp-0x4]
0x0000000000400d84 <+86>:    movsxd rdx,eax
0x0000000000400d87 <+89>:    lea    rax,[rip+0x2db682]        # 0x6dc410 <password>
0x0000000000400d8e <+96>:    mov    BYTE PTR [rdx+rax*1],cl
--Type <return> to continue, or q <return> to quit---
0x0000000000400d91 <+99>:    add    DWORD PTR [rbp-0x4],0x1
0x0000000000400d95 <+103>:   cmp    DWORD PTR [rbp-0x4],0xb
0x0000000000400d99 <+107>:   jle    0x400d43 <validate_password+21>
0x0000000000400d9b <+109>:   mov    rax,QWORD PTR [rbp-0x18]
```

Step 5: By knowing password1 and password2, password can be recovered. Password is nevergiveup!

Try this passcode.

```
(gdb) run challenge nevergiveup!
Starting program: /home/student/challenge challenge nevergiveup!
warning: Error disabling address space randomization: Operation not permitted
Success. Flag: 3e56093072782d503d54cde6e2df2901
During startup program exited normally.
(gdb)
```

Flag: 3e56093072782d503d54cde6e2df2901

References:

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