Find The Easy Pass

10th 1 23 / Document No. D22.102.100

Prepared By: clubby789

Challenge Author: Thiseas

Difficulty: Easy

Classification: Official

Synopsis

Find The Easy Pass is an easy Reversing challenge. Players will use dynamic analysis to uncover a comparison against a password.

Skills Required

- Basic disassembly skills
- String cross-referencing

Skills Learned

- Dynamically analyzing Windows applications using Wine

Solution

If we run the challenge, a window appears prompting us to enter a password. If we enter any random string and check it, we get a failure message.

If we open the binary in a decompiler, it contains a lot of unusual code patterns. Many of the strings we can see here ('Enter Password', 'Check Password') appear in the data, but don't have any cross-references. However, there *is* a cross-reference to "Wrong Password!".

```
[ .. SNIP .. ]
00454118  call    sub_40459c
0045411d  lea     edx, [ebp-0x28]
00454120  mov     eax, dword [ebx+0x2f8]
00454126  call    sub_433110
0045412b  mov     eax, dword [ebp-0x28]
0045412e  mov     edx, dword [ebp-0x4]
00454131  call    do_check
00454136  jne     0x454144

00454138  mov     eax, congrats  {"Good Job. Congratulations"}
0045413d  call    showmsgbox
00454142  jmp     0x45414e

00454144  mov     eax, password  {"Wrong Password!"}
00454149  call    showmsgbox
[ .. SNIP .. ]
```

`do_check` and `showmsgbox` here have been manually named based on what their behavior appears to be. `eax` and `edx` are set before `do_check` is called - if the function determines that they are equal, 'Good Job. Congratulations' is displayed. Otherwise, 'Wrong Password!' is displayed.

Debugging

We'll use a debugger at this point to examine the behavior of this part of the code. While many Windows debuggers are available, this challenge runs correctly under Wine in Linux, so I'll demonstrate use of `winedbg` to avoid use of paid software.

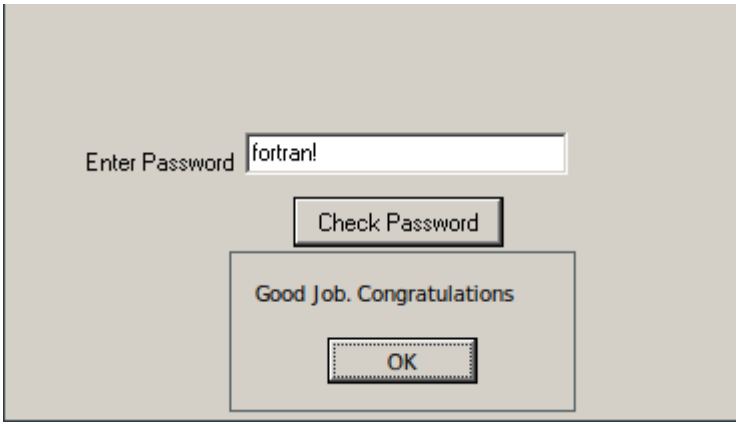
If we run `winedbg --gdb EasyPass.exe`, the binary will be started and an instance of GDB will be launched to debug it. We'll place a breakpoint before the check:

```
gef➤  b *0x0454131
Breakpoint 1 at 0x454131
gef➤  c
Continuing.
```

The password prompt appears, and we'll enter 'password' in order to reach the check. We can assume that `eax` and `edx` are passed into the function, so we'll inspect them.

```
gef➤  x/s $eax
0x1772470:  "password"
gef➤  x/s $edx
0x1773698:  "fortran!"
```

It seems like it expects the password `fortran!` - we'll try again with this.



With this, we've discovered the correct password. We can add `HTB{}` and we have the flag.