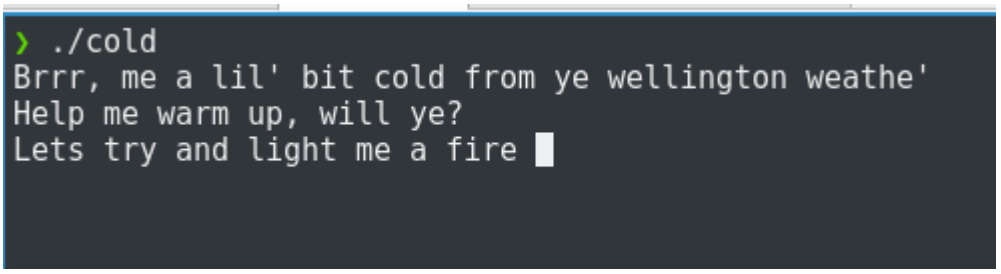
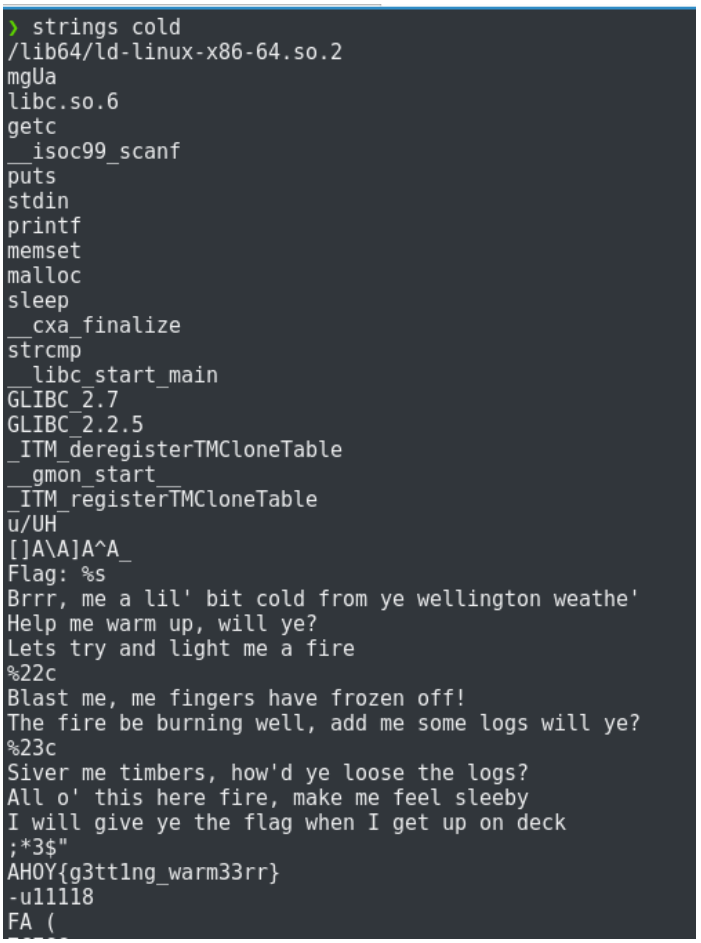
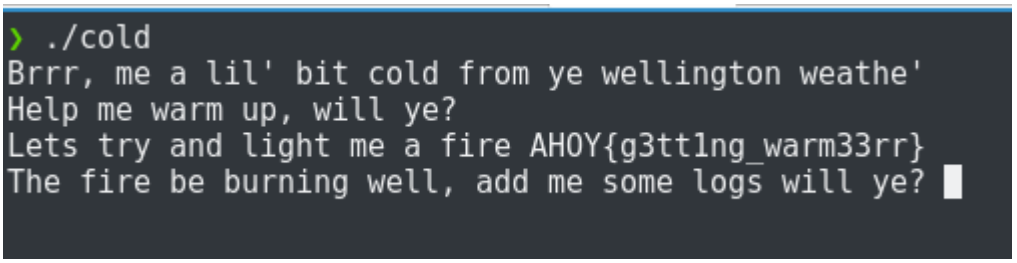
We are given a warm up binary.
Since this is an easy challenge, I will make the writeup much more descriptive.
Initially when we run the binary, we get



To take the simplest route, let's just run strings on the binary and see what we get



We can see the flag in the output.
Flag1: AHOY{g3tt1ng_warm33rr}
Trying this flag in the binary, we can see that it works



Now that we have the first flag, let's move onto the second.
Let's open up the file in ghidra and see what we get

```
Decompile: main - (cold)

undefined8 main(void)

{
  int iVar1;
  undefined8 uVar2;
  int local_c;

  puts("Brrr, me a lil\' bit cold from ye wellington weathe\'\nHelp me warm up, will ye?");
  printf("Lets try and light me a fire ");
  __isoc99_scanf(&DAT_00102084,user_input);
  getc(stdin);
  iVar1 = strcmp(user_input,first_flag);
  if (iVar1 == 0) {
    printf("The fire be burning well, add me some logs will ye? ");
    __isoc99_scanf(&DAT_001020ed,user_input);
    for (local_c = 0; local_c < 0x16; local_c = local_c + 1) {
      if (((uint)(byte)xor_string[local_c] ^ (int)(char)user_input[local_c]) != 0x45) {
        printf("Siver me timbers, how\'d ye loose the logs?");
        return 0xffffffff;
      }
    }
    puts("All o\' this here fire, make me feel sleeby");
    puts("I will give ye the flag when I get up on deck");
    sleep(0xffffffff);
    give_flag();
    uVar2 = 0;
  }
  else {
    puts("Blast me, me fingers have frozen off!");
    uVar2 = 0xffffffff;
  }
  return uVar2;
}
```

We can identify three items of interest.

- The scanf
- The strcmp
- The for loop

Let's look at the scanf first.
Looking into the first argument, for the format of the input, we get

```
            72 79 20 ...

                         DAT_00102084                                    XREF[1]:
00102084 25          ??          25h    %
00102085 32          ??          32h    2
00102086 32          ??          32h    2
00102087 63          ??          63h    c
00102088 00          ??          00h
00102089 00          ??          00h
0010208a 00          ??          00h
0010208b 00          ??          00h
0010208c 00          ??          00h
0010208d 00          ??          00h
0010208e 00          ??          00h
```

Which is %22c.
Researching a bit, we can see that it is getting 22 characters.

Looking now onto the strcmp,
We can see that there is a strcmp onto the *user_input* and the *first_flag*

```
printf("Lets try and light me a fire ");
__isoc99_scanf(&char_format,user_input);
getc(stdin);
iVar1 = strcmp(user_input,first_flag);
if (iVar1 == 0) {
```

Looking into the first_flag argument, we can see

```
                        first_flag                                      XREF[
00104070 41 48 4f        undefine...
         59 7b 67
         33 74 74 ...
   00104070 41              undefined141h                  [0]
   00104071 48              undefined148h                  [1]
   00104072 4f              undefined14Fh                  [2]
   00104073 59              undefined159h                  [3]
   00104074 7b              undefined17Bh                  [4]
   00104075 67              undefined167h                  [5]
   00104076 33              undefined133h                  [6]
   00104077 74              undefined174h                  [7]
   00104078 74              undefined174h                  [8]
   00104079 31              undefined131h                  [9]
   0010407a 6e              undefined16Eh                  [10]
   0010407b 67              undefined167h                  [11]
   0010407c 5f              undefined15Fh                  [12]
   0010407d 77              undefined177h                  [13]
   0010407e 61              undefined161h                  [14]
   0010407f 72              undefined172h                  [15]
   00104080 6d              undefined16Dh                  [16]
   00104081 33              undefined133h                  [17]
   00104082 33              undefined133h                  [18]
   00104083 72              undefined172h                  [19]
   00104084 72              undefined172h                  [20]
   00104085 7d              undefined17Dh                  [21]
   00104086 00              undefined100h                  [22]
00104087 00              ??         00h
```

Which isn't particularly useful, however changing the format to a char[22] we get

```
                        first_flag
00104070 41 48 4f        char[22]    "AHOY{g3ttlng_warm33rr}"
         59 7b 67
         33 74 74 ...
   00104070 [0]             'A', 'H', 'O', 'Y'
   00104074 [4]             '{', 'g', '3', 't'
   00104078 [8]             't', 'l', 'n', 'g'
   0010407c [12]            '_', 'w', 'a', 'r'
   00104080 [16]            'm', '3', '3', 'r'
   00104084 [20]            'r', '}'
00104086 00              33         00h
```

Which was that same flag that we found earlier using strings.
If we look at the output of the strcmp, we can that it is checking for a return value of zero

```
iVar1 = strcmp(user_input,first_flag);
if (iVar1 == 0) {
  printf("The fire be burning well, add me some logs will ye? ");
  __isoc99_scanf(&DAT_001020ed,user_input);
  for (local_c = 0; local_c < 0x16; local_c = local_c + 1) {
    if (((uint)(byte)xor_string[local_c] ^ (int)(char)user_input[local_c]) != 0x45) {
      printf("Siver me timbers, how\'d ye loose the logs?");
      return 0xffffffff;
    }
```

If we now look into the for loop, we can see an if statement checking if an xor is not equal to 0x45, and exiting if they are not equal.
Cleaning up the for loop a little bit

```
__isoc99_scanf(&DAT_001020ed,user_input);
for (counter = 0; counter < 0x16; counter = counter + 1) {
  if (((uint)(byte)xor_string[counter] ^ (int)(char)user_input[counter]) != 0x45) {
    printf("Siver me timbers, how\'d ye loose the logs?");
    return 0xffffffff;
  }
}
```

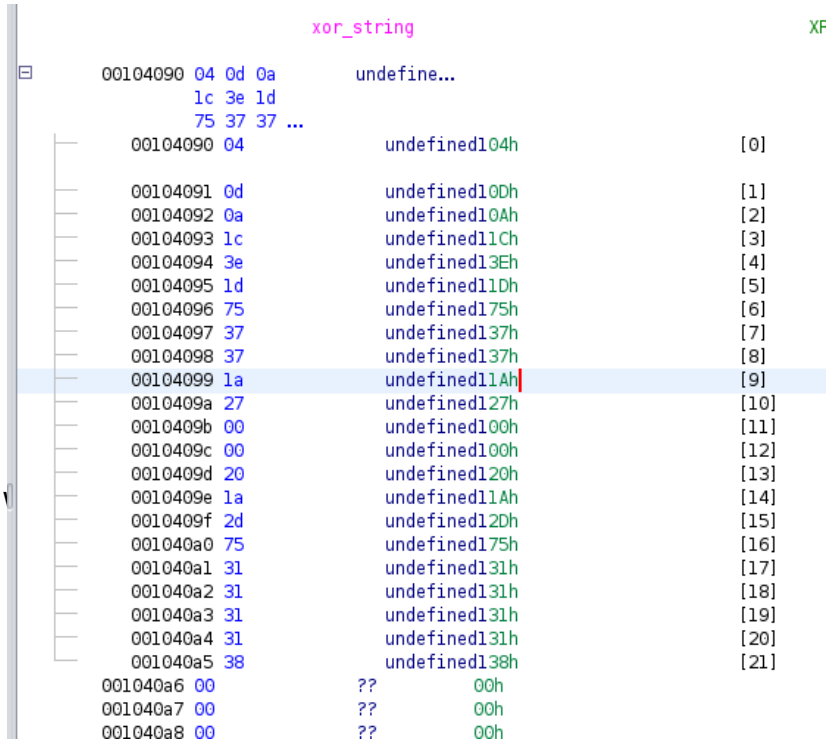
We can see the binary checking for

xor_string[i] ^ user_input[i] == 0x45

And looking for 22 characters

One thing about xor, is that it can undo itself so we can have
user_input[i] = 0x45 ^ xor_string[i]

If we dump xor_string, which currently looks like



```
❯ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> xor_string = [ '\x04','\r','\n', '\x1C','>', '\x1D', 'u', '7','7', '\x1A','\'','\0','\0', ' ', '\x1A', '-', 'u', '1', '1', '1','1', '8']
>>> user_input = [chr(ord(i) ^ 0x45) for i in xor_string]
>>> ''.join(user_input)
'AHOY{X0rr_bEEe_h0tttt}'
>>>
```

We can see that we get a flag of
AHOY{X0rr_bEEe_h0tttt}

Putting these two flags into the binary and we get

```
❯ ./cold
Brrr, me a lil' bit cold from ye wellington weathe'
Help me warm up, will ye?
Lets try and light me a fire AHOY{g3tt1ng_warm33rr}
The fire be burning well, add me some logs will ye? AHOY{X0rr_bEEe_h0tttt}
All o' this here fire, make me feel sleeby
I will give ye the flag when I get up on deck
```

For the third flag, looking into the binary we get

```
    puts("All o\' this here fire, make me feel sleeby");
    puts("I will give ye the flag when I get up on deck");
    sleep(0xffffffff);
    give_flag();
    uVar2 = 0;
```

Two puts a sleep for 0xffffffff and then the give flag function, looking at the sleep function with *man 3 sleep*

```
SLEEP(3)

NAME
       sleep - sleep for a specified number of seconds

SYNOPSIS
       #include <unistd.h>

       unsigned int sleep(unsigned int seconds);

DESCRIPTION
       sleep() causes the calling thread to sleep either until
```

We can see that this program would sleep for an extremely long time before giving us the flag.

Looking into the function itself

Decompile: give_flag - (cold)

```
1
2  void give_flag(void)
3
4  {
5    void *__s;
6    int local_c;
7
8    __s = malloc(0x17);
9    memset(__s,0,0x17);
10   for (local_c = 0; local_c < 0x16; local_c = local_c + 1) {
11     *(undefined1 *)((long)__s + (long)local_c) = user_input[local_c] ^ final_flag[local_c];
12   }
13   printf("Flag: %s\n",__s);
14   return;
15 }
16
```

We can see that it performs an xor with a loop and prints out the flag and the key thing is that it doesn't seem to take any arguments.
However to improving the decompilation we can recognise the counter and the fact that the malloc call should be a char *, we get

Decompile: give_flag - (cold)

```
1
2  void give_flag(void)
3
4  {
5    char *flag;
6    int counter;
7
8    flag = (char *)malloc(0x17);
9    memset(flag,0,0x17);
10   for (counter = 0; counter < 0x16; counter = counter + 1) {
11     flag[counter] = user_input[counter] ^ final_flag[counter];
12   }
13   printf("Flag: %s\n",flag);
14   return;
15 }
16
```

Currently now, we have two possible options of getting the flag

- We could patch out the sleep instruction, allowing the binary to print the flag for us
- Or while debugging, jump over the call to sleep

I will show both options, starting with the debugging option

Running the cold binary in gdb we have

```
❯ gdb ./cold
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 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.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from ./cold...(no debugging symbols found)...done.
pwndbg>
```

Also to note, I use the gdb “plugin” which is at https://github.com/pwndbg/pwndbg, which I feel makes the debugging experience much nicer than standard gdb

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 198 commands. Type pwndbg [filter] for a list.
pwndbg: created $rebase, $ida gdb functions (can be used with print/break)
Reading symbols from ./cold...(no debugging symbols found)...done.
pwndbg> b main
Breakpoint 1 at 0x1236
pwndbg> r
Starting program: /home/ava/Projects/vec_ctf_2/warmup_rev/cold

Breakpoint 1, 0x0000555555555236 in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
 RAX  0x555555555232 (main) ◂— push   rbp
 RBX  0x0
 RCX  0x7ffff7fa6718 (__exit_funcs) —▸ 0x7ffff7fa7d80 (initial) ◂— 0x0
 RDX  0x7fffffffe038 —▸ 0x7fffffffe3c5 ◂— 'BROWSER=firefox'
 RDI  0x1
 RSI  0x7fffffffe028 —▸ 0x7fffffffe398 ◂— '/home/ava/Projects/vec_ctf_2/warmup_rev/cold'
 R8   0x7ffff7fa7d80 (initial) ◂— 0x0
 R9   0x7ffff7fa7d80 (initial) ◂— 0x0
 R10  0x3
 R11  0x2
 R12  0x5555555550c0 (_start) ◂— xor    ebp, ebp
 R13  0x7fffffffe020 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdf40 —▸ 0x555555555370 (__libc_csu_init) ◂— push   r15
 RSP  0x7fffffffdf40 —▸ 0x555555555370 (__libc_csu_init) ◂— push   r15
 RIP  0x555555555236 (main+4) ◂— sub    rsp, 0x20
[ DISASM ]
► 0x555555555236 <main+4>     sub    rsp, 0x20
  0x55555555523a <main+8>     mov    dword ptr [rbp - 0x14], edi
  0x55555555523d <main+11>    mov    qword ptr [rbp - 0x20], rsi
  0x555555555241 <main+15>    lea    rdi, [rip + 0xdd0]
  0x555555555248 <main+22>    call   puts@plt                <puts@plt>

  0x55555555524d <main+27>    lea    rdi, [rip + 0xe12]
  0x555555555254 <main+34>    mov    eax, 0
  0x555555555259 <main+39>    call   printf@plt                <printf@plt>

  0x55555555525e <main+44>    lea    rsi, [rip + 0x2e7b]           <0x5555555580e0>
  0x555555555265 <main+51>    lea    rdi, [rip + 0xe18]
  0x55555555526c <main+58>    mov    eax, 0
[ STACK ]
00:0000| rbp rsp 0x7fffffffdf40 —▸ 0x555555555370 (__libc_csu_init) ◂— push   r15
01:0008|         0x7fffffffdf48 —▸ 0x7ffff7e1009b (__libc_start_main+235) ◂— mov    edi, eax
02:0010|         0x7fffffffdf50 ◂— 0x0
03:0018|         0x7fffffffdf58 —▸ 0x7fffffffe028 —▸ 0x7fffffffe398 ◂— '/home/ava/Projects/vec_ctf_2/warmup_rev/cold'
04:0020|         0x7fffffffdf60 ◂— 0x100040000
05:0028|         0x7fffffffdf68 —▸ 0x555555555232 (main) ◂— push   rbp
06:0030|         0x7fffffffdf70 ◂— 0x0
07:0038|         0x7fffffffdf78 ◂— 0x56dc206b0fd2140
[ BACKTRACE ]
► f 0   0x555555555236 main+4
  f 1   0x7ffff7e1009b __libc_start_main+235

pwndbg>
```

Putting the commands
b main
r
Puts a breakpoint at main and starts the binary

Now you might be thinking to yourself,
“Ava, can’t we just go the function first”
Well the problem with that is looking back at the decompilation, it requires

Decompile: give_flag - (cold)

```
1
2  void give_flag(void)
3
4  {
5    char *flag;
6    int counter;
7
8    flag = (char *)malloc(0x17);
9    memset(flag,0,0x17);
10   for (counter = 0; counter < 0x16; counter = counter + 1) {
11     flag[counter] = user_input[counter] ^ final_flag[counter];
12   }
13   printf("Flag: %s\n",flag);
14   return;
15 }
16
```

User_input to be set, so we will have to break just before the sleep, skip over and continue

Going back to gdb, we can disassemble with the command
near pc 33
Which is very similar to the *disassemble* command, except that it has nicer output.

```
   0x5555555552ad <main+123>          jmp    main+306               <main+306>

   0x5555555552b2 <main+128>          lea    rdi, [rip + 0xdff]
   0x5555555552b9 <main+135>          mov    eax, 0
   0x5555555552be <main+140>          call   printf@plt               <printf@plt>

   0x5555555552c3 <main+145>          lea    rsi, [rip + 0x2e16]          <0x5555555580e0>
   0x5555555552ca <main+152>          lea    rdi, [rip + 0xe1c]
   0x5555555552d1 <main+159>          mov    eax, 0
   0x5555555552d6 <main+164>          call   __isoc99_scanf@plt               <__isoc99_scanf@plt>

   0x5555555552db <main+169>          mov    dword ptr [rbp - 4], 0
   0x5555555552e2 <main+176>          jmp    main+251               <main+251>

   0x5555555552e4 <main+178>          mov    eax, dword ptr [rbp - 4]
   0x5555555552e7 <main+181>          cdqe
   0x5555555552e9 <main+183>          lea    rdx, [rip + 0x2df0]          <0x5555555580e0>
   0x5555555552f0 <main+190>          movzx  eax, byte ptr [rax + rdx]
   0x5555555552f4 <main+194>          movsx  edx, al
   0x5555555552f7 <main+197>          mov    eax, dword ptr [rbp - 4]
   0x5555555552fa <main+200>          cdqe
   0x5555555552fc <main+202>          lea    rcx, [rip + 0x2d8d]          <0x555555558090>
   0x555555555303 <main+209>          movzx  eax, byte ptr [rax + rcx]
   0x555555555307 <main+213>          movzx  eax, al
   0x55555555530a <main+216>          xor    eax, edx
   0x55555555530c <main+218>          cmp    eax, 0x45
   0x55555555530f <main+221>          je     main+247               <main+247>

   0x555555555311 <main+223>          lea    rdi, [rip + 0xde0]
   0x555555555318 <main+230>          mov    eax, 0
   0x55555555531d <main+235>          call   printf@plt               <printf@plt>

   0x555555555322 <main+240>          mov    eax, 0xffffffff
   0x555555555327 <main+245>          jmp    main+306               <main+306>

   0x555555555329 <main+247>          add    dword ptr [rbp - 4], 1
   0x55555555532d <main+251>          cmp    dword ptr [rbp - 4], 0x15
   0x555555555331 <main+255>          jle    main+178               <main+178>

   0x555555555333 <main+257>          lea    rdi, [rip + 0xdee]
   0x55555555533a <main+264>          call   puts@plt                <puts@plt>

   0x55555555533f <main+269>          lea    rdi, [rip + 0xe12]
   0x555555555346 <main+276>          call   puts@plt                <puts@plt>

   0x55555555534b <main+281>          mov    edi, 0xffffffff
   0x555555555350 <main+286>          call   sleep@plt               <sleep@plt>

   0x555555555355 <main+291>          mov    eax, 0
   0x55555555535a <main+296>          call   give_flag               <give_flag>

   0x55555555535f <main+301>          mov    eax, 0
   0x555555555364 <main+306>          leave
   0x555555555365 <main+307>          ret
```

Pwndbg has a nice feature that disables ASLR, which for this case doesn't really mean anything, except means that the memory layout should normally be the same.
Looking at this disassembly, we can see the

```
0x55555555533f <main+269>          lea    rdi, [rip + 0xe12]
0x555555555346 <main+276>          call   puts@plt                <puts@plt>

0x55555555534b <main+281>          mov    edi, 0xffffffff
0x555555555350 <main+286>          call   sleep@plt                <sleep@plt>

0x555555555355 <main+291>          mov    eax, 0
0x55555555535a <main+296>          call   give_flag                <give_flag>
```

Which calls put, sleeps and then calls give_flag
What we can do is but a breakpoint on

```
0x55555555534b <main+281>          mov    edi, 0xffffffff
```

And then once we hit it, we can jump over to

```
0x555555555355 <main+291>          mov    eax, 0
```

Which is right after the sleep call

So putting a breakpoint in gdb with
*b *0x55555555534b*

And then continuing with
c

We will need to put in both flags to get up to that point, once we hit it, we get

```
Brrr, me a lil' bit cold from ye wellington weathe'
Help me warm up, will ye?
Lets try and light me a fire AHOY{g3tt1ng_warm33rr}
The fire be burning well, add me some logs will ye? AHOY{X0rr_bEEe_h0tttt}
All o' this here fire, make me feel sleeby
I will give ye the flag when I get up on deck

Breakpoint 2, 0x000055555555534b in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[ REGISTERS ]
*RAX  0x2e
 RBX  0x0
*RCX  0x7ffff7ed6274 (write+20) ◂— cmp    rax, -0x1000 /* 'H=' */
*RDX  0x7ffff7fa88c0 (_IO_stdfile_1_lock) ◂— 0x0
*RDI  0x0
*RSI  0x5555555559260 ◂— 'I will give ye the flag when I get up on deck\nl ye? '
*R8   0x7ffff7fad500 ◂— 0x7ffff7fad500
*R9   0x20
*R10  0x5555555560ed ◂— 0x63333225 /* '%23c' */
*R11  0x246
 R12  0x5555555550c0 (_start) ◂— xor    ebp, ebp
 R13  0x7fffffffe020 ◂— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdf40 —▸ 0x555555555370 (__libc_csu_init) ◂— push   r15
*RSP  0x7fffffffdf20 —▸ 0x7fffffffe028 —▸ 0x7fffffffe398 ◂— '/home/ava/Projects/vec_ctf_2/warmup_rev/cold'
*RIP  0x55555555534b (main+281) ◂— mov    edi, 0xffffffff
[ DISASM ]
   0x555555555331 <main+255>    jle    main+178                <main+178>

   0x555555555333 <main+257>    lea    rdi, [rip + 0xdee]
   0x55555555533a <main+264>    call   puts@plt                <puts@plt>

   0x55555555533f <main+269>    lea    rdi, [rip + 0xe12]
   0x555555555346 <main+276>    call   puts@plt                <puts@plt>

 ► 0x55555555534b <main+281>    mov    edi, 0xffffffff
   0x555555555350 <main+286>    call   sleep@plt                <sleep@plt>

   0x555555555355 <main+291>    mov    eax, 0
   0x55555555535a <main+296>    call   give_flag                <give_flag>

   0x55555555535f <main+301>    mov    eax, 0
   0x555555555364 <main+306>    leave
[ STACK ]
00:0000| rsp 0x7fffffffdf20 —▸ 0x7fffffffe028 —▸ 0x7fffffffe398 ◂— '/home/ava/Projects/vec_ctf_2/warmup_rev/cold'
01:0008|     0x7fffffffdf28 ◂— 0x1555550c0
02:0010|     0x7fffffffdf30 —▸ 0x7fffffffe020 ◂— 0x1
03:0018|     0x7fffffffdf38 ◂— 0x1600000000
04:0020| rbp 0x7fffffffdf40 —▸ 0x555555555370 (__libc_csu_init) ◂— push   r15
05:0028|     0x7fffffffdf48 —▸ 0x7ffff7e1009b (__libc_start_main+235) ◂— mov    edi, eax
06:0030|     0x7fffffffdf50 ◂— 0x0
07:0038|     0x7fffffffdf58 —▸ 0x7fffffffe028 —▸ 0x7fffffffe398 ◂— '/home/ava/Projects/vec_ctf_2/warmup_rev/cold'
[ BACKTRACE ]
 ► f 0   0x55555555534b main+281
   f 1   0x7ffff7e1009b __libc_start_main+235

pwndbg>
```

To set the instruction pointer, a register that is used to tell the cpu what instruction to execute, we can use the command (after googling)
how to set rip gdb

How can i set the variable rip in "info registers" in GDB console? [duplicate

Asked 3 years, 1 month ago Modified 3 years, 1 month ago Viewed 3k times

3

This question already has answers here:
Is it possible to "jump"/"skip" in GDB debugger? (2 answers)
Closed 3 years ago.

How can i set RIP in "info registers" of a program with gdb? Like you do for eax, you type "set $eax=0", but how can i do for rip?

assembly binary gdb

Share Improve this question Follow

edited Jul 12, 2019 at 9:16
ks1322
31.9k • 13 • 100 • 155

asked Jul 11, 2019 at 20:34
c0bra
31 • 1 • 5

2 Yes, `set $pc = 0x123456`. RIP isn't a "normal" register, so GDB treats it specially using the general "program counter" name regardless of architecture. – Peter Cordes Jul 12, 2019 at 9:26

Add a comment

We can see that the command is
set $pc = value

Looking back onto the disassembly, we can see that we want

```
► 0x55555555534b <main+281>    mov    edi, 0xffffffff
  0x555555555350 <main+286>    call   sleep@plt                <sleep@plt>

  0x555555555355 <main+291>    mov    eax, 0
  0x55555555535a <main+296>    call   give_flag                <give_flag>

  0x55555555535f <main+301>    mov    eax, 0
  0x555555555364 <main+306>    leave
```

rip or pc, to be set to 0x555555555355,

So with the command
set $pc = 0x555555555355

```
  0x55555555534b <main+281>    mov    edi, 0xffffffff
  0x555555555350 <main+286>    call   sleep@plt                <sleep@plt>

► 0x555555555355 <main+291>    mov    eax, 0
  0x55555555535a <main+296>    call   give_flag                <give_flag>

  0x55555555535f <main+301>    mov    eax, 0
  0x555555555364 <main+306>    leave
```

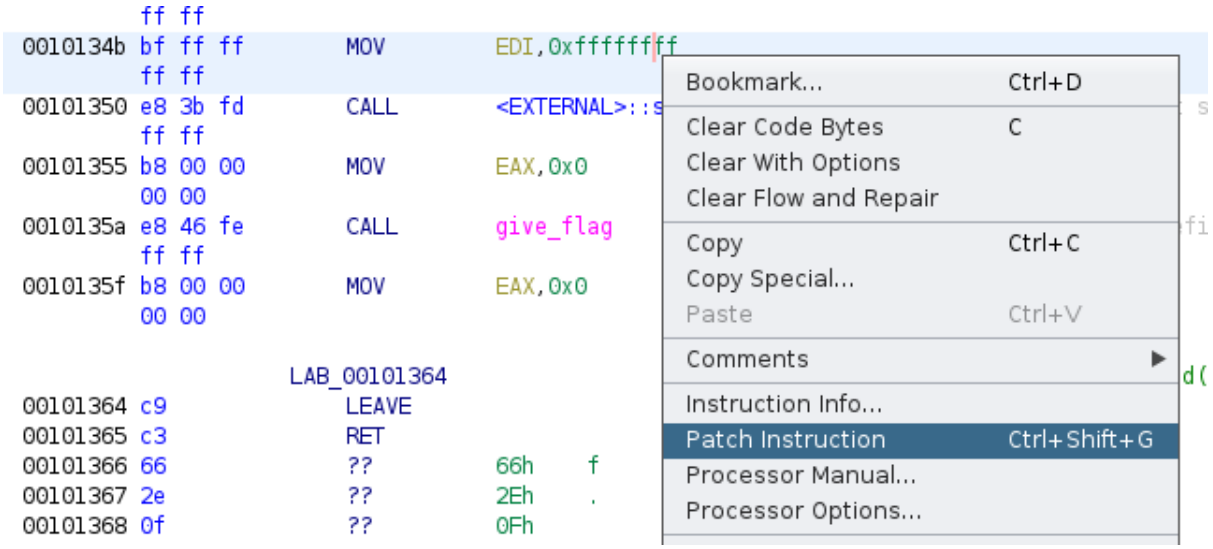
We can see that we jumped over the sleep call, continuing with the command
c

We get

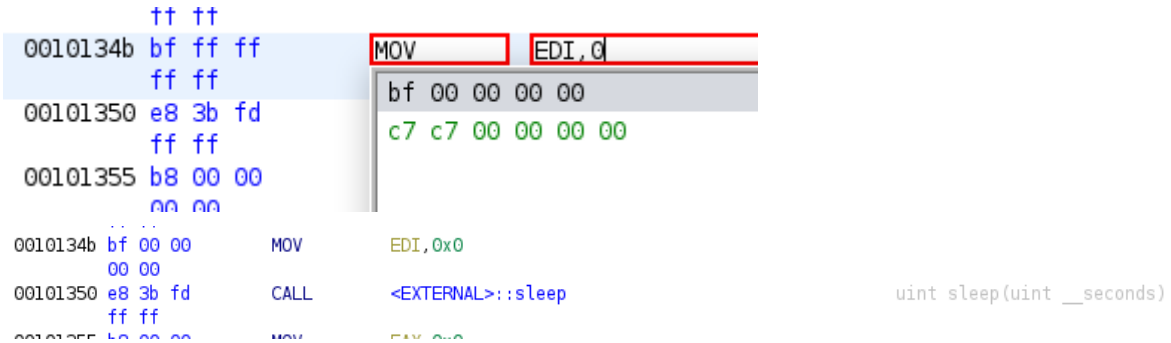
```
pwndbg> c
Continuing.
Flag: AHOY{0v3Rwr1T3__s133p}
[Inferior 1 (process 3102) exited normally]
pwndbg>
```

Which is the final flag

Now for the alternative way, we can do the patching method
Going back to ghidra
If we right click and then go to *patch instruction*

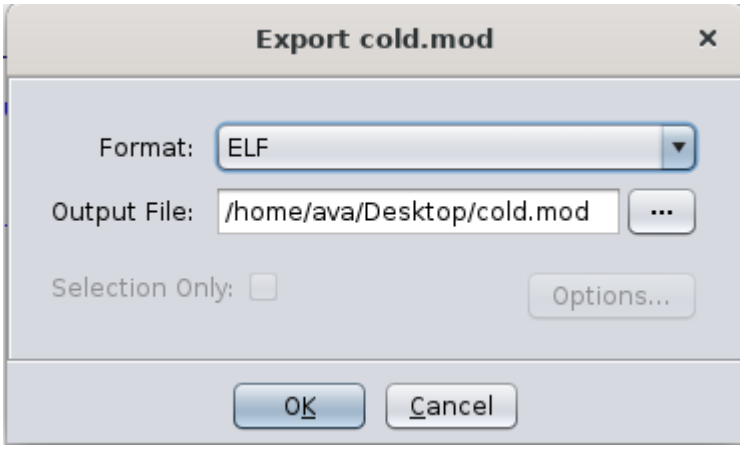


We can then change the value that is being passed to zero



Which will cause the program to sleep for zero seconds.
You could also change it so that the mov + call was completely noped out (NOP is an No OPeration instruction which basically does nothing), but I think this is a cool a simple way

We haven't actually changed the binary on disk instead we have changed the ghidra one, so we will need to export the binary back to disk so we can run it.



Running the modified cold with the flags given, we have

```
❯ ./cold_elf.mod
Brrr, me a lil' bit cold from ye wellington weathe'
Help me warm up, will ye?
Lets try and light me a fire AHOY{g3tt1ng_warm33rr}
The fire be burning well, add me some logs will ye? AHOY{X0rr_bEEe_h0tttt}
All o' this here fire, make me feel sleeby
I will give ye the flag when I get up on deck
Flag: AHOY{Ov3Rwr1T3__s133p}
```

Which gives us the flag