

iii) GDB Baby Step 1:

The following is the question:

GDB baby step 1

| 100 points

Tags:

picoGym Exclusive

Reverse Engineering

x86_64

AUTHOR: LT 'SYREAL' JONES

Description

Can you figure out what is in the `eax` register at the end of the `main` function? Put your answer in the picoCTF flag format: `picoCTF{n}` where `n` is the contents of the `eax` register in the decimal number base. If the answer was `0x11` your flag would be `picoCTF{17}`.
Disassemble [this](#).

Hints ?
1 2

1,441 solves / 1,485 users attempted (97%)

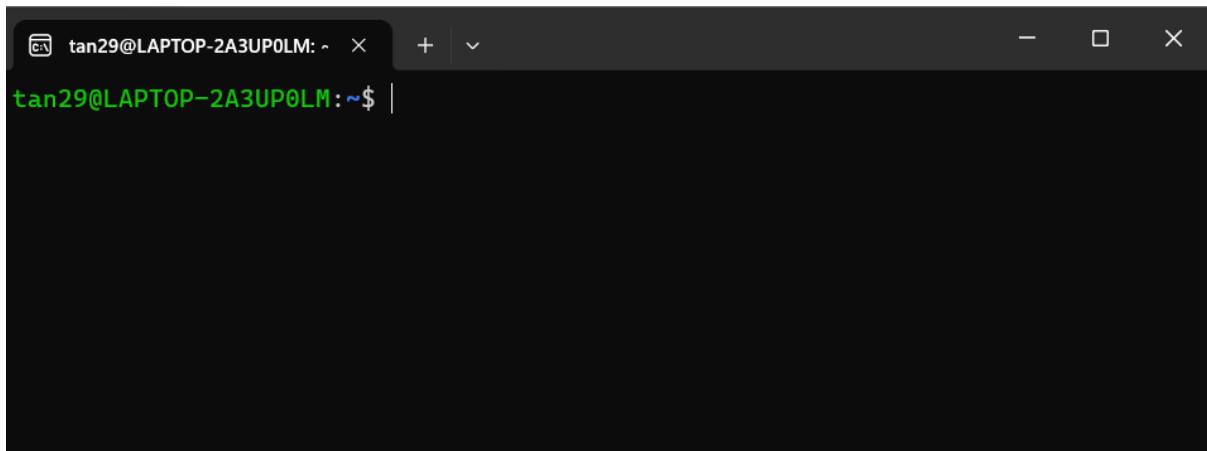
71%
Liked

picoCTF{FLAG}

Submit Flag

Upon viewing the hints, it is essential that gdb (a portable debugger that supports many languages and is designed for Unix based systems) is to be used to find the bugs in the needed code.

We download the file and then open the terminal.

A screenshot of a terminal window. The title bar shows 'tan29@LAPTOP-2A3UP0LM: ~' with standard window controls. The terminal content shows a green prompt 'tan29@LAPTOP-2A3UP0LM: ~\$' followed by a vertical bar cursor.

(this is WSL, not VMWare which I used in the previous write-up)

We install gdb using command:

```
sudo apt install gdb
```

sudo	Short for “super user do”. There are many functions a normal user is not allowed to perform and it is not always safe to be operating as the root user. So we use the sudo command to perform some functions we wouldn’t be able to do normally.
apt	Short for “Advanced Package Tool”, a software that helps us to install, remove and update packages on Debian(open

	source computer OS system) based OS such as Ubuntu
install	A simple install command which will install the package we specify ahead of it
gdb	The package we want to install at present

```
tan29@LAPTOP-2A3UP0LM:~$ sudo apt install gdb
[sudo] password for tan29:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libbabeltrace1 libboost-regex1.74.0 libc6-dbg libdebuginfod-common
  libdebuginfod1 libipt2 libsource-highlight-common libsource-highlight4v5
Suggested packages:
  gdb-doc gdbserver
The following NEW packages will be installed:
  gdb libbabeltrace1 libboost-regex1.74.0 libc6-dbg libdebuginfod-common
  libdebuginfod1 libipt2 libsource-highlight-common libsource-highlight4v5
0 upgraded, 9 newly installed, 0 to remove and 0 not upgraded.
Need to get 18.7 MB of archives.
After this operation, 35.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

After confirming we wait for some time and then gdb is installed.

To enter gdb, we simply type 'gdb' in the terminal:

```
tan29@LAPTOP-2A3UP0LM: ~$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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:
<https://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".
(gdb) |
```

Since we need to view the flow of the program in order to find our flag, this is what we do:

The file command looks for bugs in our file while simultaneously executing it, and returns the bugs (if any). So we run the 'file' command.

```
(gdb) file debugger0_a
Reading symbols from debugger0_a...
(No debugging symbols found in debugger0_a)
```

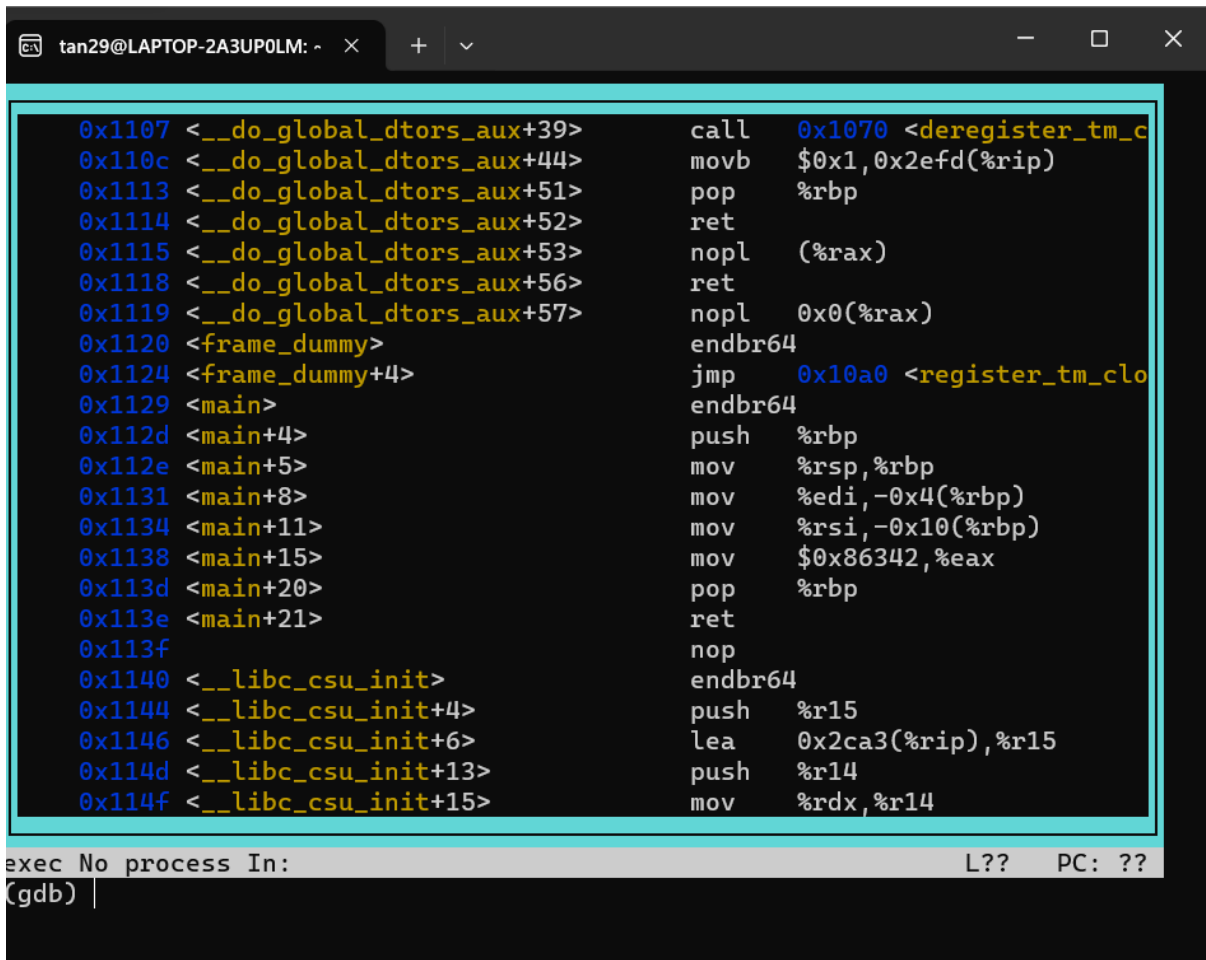
It shows us that no debugging symbols(They map instructions in the compiled binary program to their corresponding variable, function, or line in the source code) were found.

After some reading online, I found that every code is broken down to assembly language first. So I decided to switch to ASM environment in gdb.

To do so I used the 'layout asm' command,

```
(gdb) file debugger0_a
Reading symbols from debugger0_a...
(No debugging symbols found in debugger0_a)
(gdb) layout asm
```

This is the screen displayed:



```
tan29@LAPTOP-2A3UP0LM: ~ × + ▾
0x1107 <__do_global_dtors_aux+39>    call    0x1070 <deregister_tm_c
0x110c <__do_global_dtors_aux+44>    movb    $0x1,0x2efd(%rip)
0x1113 <__do_global_dtors_aux+51>    pop     %rbp
0x1114 <__do_global_dtors_aux+52>    ret
0x1115 <__do_global_dtors_aux+53>    nopl    (%rax)
0x1118 <__do_global_dtors_aux+56>    ret
0x1119 <__do_global_dtors_aux+57>    nopl    0x0(%rax)
0x1120 <frame_dummy>                endbr64
0x1124 <frame_dummy+4>              jmp     0x10a0 <register_tm_clo
0x1129 <main>                        endbr64
0x112d <main+4>                      push    %rbp
0x112e <main+5>                      mov     %rsp,%rbp
0x1131 <main+8>                      mov     %edi,-0x4(%rbp)
0x1134 <main+11>                     mov     %rsi,-0x10(%rbp)
0x1138 <main+15>                     mov     $0x86342,%eax
0x113d <main+20>                     pop     %rbp
0x113e <main+21>                     ret
0x113f                             nop
0x1140 <__libc_csu_init>            endbr64
0x1144 <__libc_csu_init+4>           push    %r15
0x1146 <__libc_csu_init+6>           lea     0x2ca3(%rip),%r15
0x114d <__libc_csu_init+13>          push    %r14
0x114f <__libc_csu_init+15>          mov     %rdx,%r14
exec No process in: L?? PC: ??
(gdb) |
```

Of this, we see

```
0x1138 <main+15>          mov     $0x86342,%eax
```

0x1138	This is the memory address where the instruction is located. It is in hexadecimal form.
<main+15>	This indicates that the instruction is 15 bytes offset from the start of a function named main. The main function is usually the entry point for execution in most programs.
mov	"mov" is a commonly used x86 Assembly instruction that copies data from one location to another.
0x86342	This represents an immediate value (a constant number) in hexadecimal that is to be moved into the destination operand.
%eax	This is the destination operand, which in this case is the eax register.

In a nutshell, at the memory address 0x1138, which is 15 bytes into the main function, there is

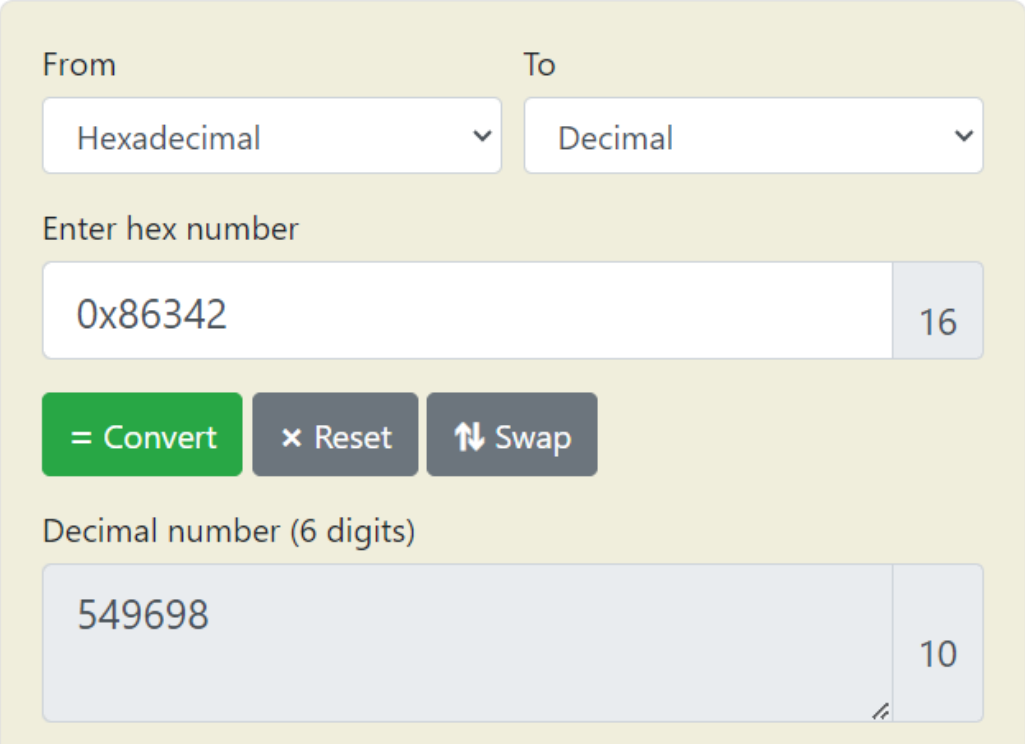
an instruction (mov) that moves the hexadecimal constant 0x86342 into the eax register.

This matches our needed format.

So 0x86342 is our needed flag.

Converting it into the needed form:

Hexadecimal to Decimal converter



The image shows a web-based converter interface. At the top, there are two dropdown menus labeled 'From' and 'To'. The 'From' menu is set to 'Hexadecimal' and the 'To' menu is set to 'Decimal'. Below these is a text input field labeled 'Enter hex number' containing the value '0x86342'. To the right of this field is a small box containing the number '16'. Below the input field are three buttons: a green button labeled '= Convert', a grey button labeled 'x Reset', and a grey button labeled '↕ Swap'. Below the buttons is a text input field labeled 'Decimal number (6 digits)' containing the value '549698'. To the right of this field is a small box containing the number '10'.

From	To
Hexadecimal	Decimal

Enter hex number

0x86342 16

= Convert x Reset ↕ Swap

Decimal number (6 digits)

549698 10

Our flag is:

picoCTF{549698}

Passing this into picoCTF:



[Sources: Stack Overflow, Apple Dev, RapidTables]