

Initial Setup Steps for Ubuntu

1. Make sure the binary is executable / runnable
 - a. To make a binary executable, Change permission to make the binary be executable either using GUI (**Right Click on the file → Properties → Permissions → check box for execute**) or using the command line (**chmod +x <path-to-file>**)
2. Make sure the correct compiler / Interpreter / Debugger is present.
 - a. The binaries are in 32-bit arch and we have a 64-bit OS. So to make sure that the binaries can be executed, install the correct debugger and compiler using (**sudo apt-get install libc6:i386**)
3. Make sure the binary is run using (**./<path to binary> in terminal**)
4. Set Disassembly flavor / style
 - a. We will be using att for our example but you can set any flavor using **set disassembly-flavor intel/att**

Binary 1 - Crackme0x00 - Password: 250382

Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **acbd and press enter**
8. The screen should like like following

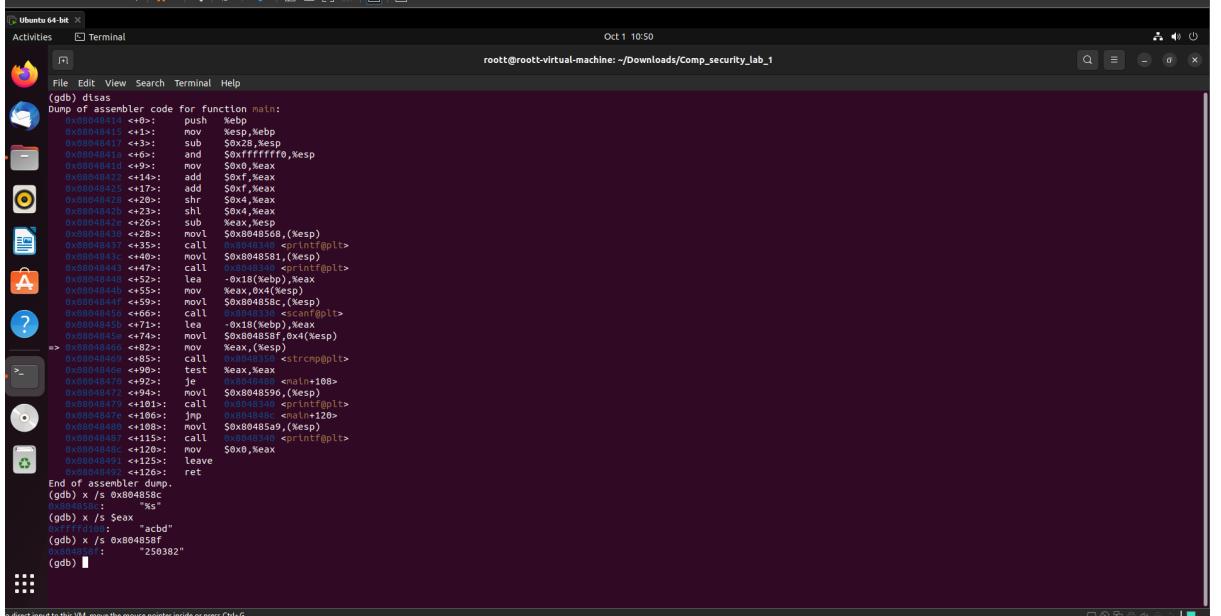
```
root@root-virtual-machine:~/Downloads/Comp_security_lab_1$ gdb ./crackme0x00
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY; for details see the warranty section of the license,
Type "show copying" and "show warranty" for details.
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"...
Reading symbols from /home/root/Downloads/Comp_security_lab_1/crackme0x00
(gdb) run
Starting program: /home/root/Downloads/Comp_security_lab_1/crackme0x00
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Using core level 0x60
Password:
Program received signal SIGINT, Interrupt.
0x00040454 in _kernel_vsyscall
(gdb) bt
#0 0x00040454 in _kernel_vsyscall ()
#1 0x00040457 in read () from /lib/x86_64-linux-gnu/libc.so.6
#2 0x00040458 in __IO_file_underflow () from /lib/x86_64-linux-gnu/libc.so.6
#3 0x00040459 in __IO_default_uflow () from /lib/x86_64-linux-gnu/libc.so.6
#4 0x0004045a in __IO_file_scanf () from /lib/x86_64-linux-gnu/libc.so.6
#5 0x0004045b in scanf () from /lib/x86_64-linux-gnu/libc.so.6
(gdb) tbreak *0x0004045b
Temporary breakpoint 1 at 0x0004045b
(gdb) c
Continuing.
acbd

Temporary breakpoint 1, 0x0004045b in main ()
(gdb)
```

9. Disassemble the binary using **disas**

10. Move to next instruction using **nexti** such that we are just before the comparison line
11. Find the input format for **scanf**
12. Find the comparison line **strcmp**
13. Find the values being compared by looking into the individual values
14. The output should look like the following screen

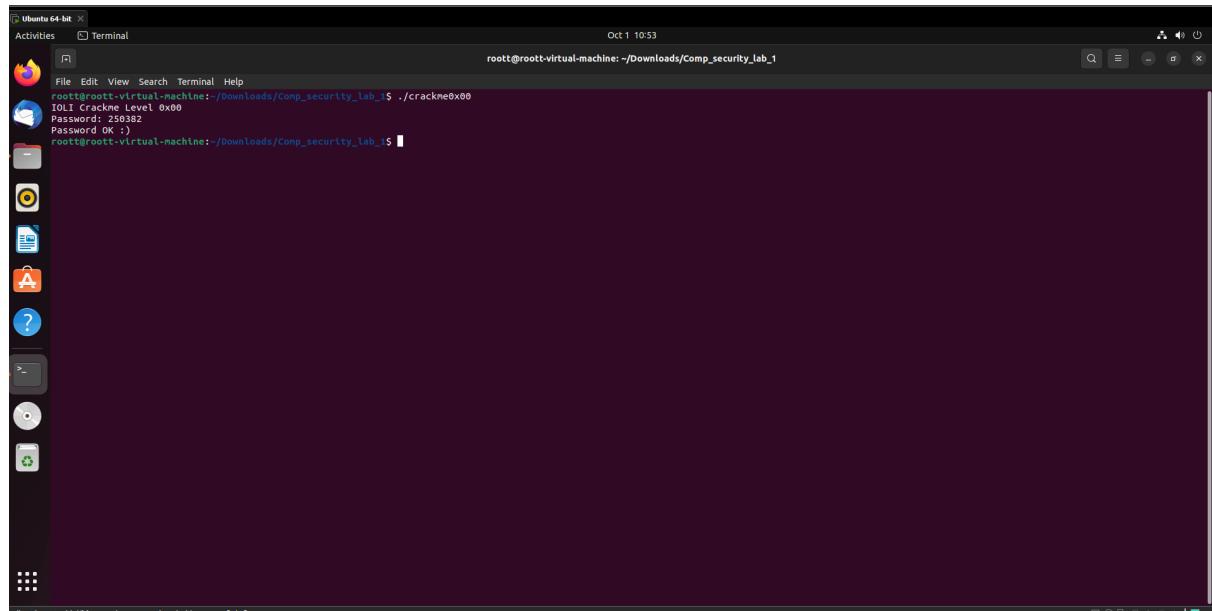


The screenshot shows the GDB debugger interface on an Ubuntu 64-bit system. The terminal window displays the assembly code for the `main` function. The assembly dump shows various instructions including pushes, adds, and comparisons. The debugger prompt is visible at the bottom.

```
(gdb) disas
Dump of assembler code for function main:
0x08048411 <+0>: push %ebp
0x08048411 <+1>: mov %esp,%ebp
0x08048414 <+2>: sub $0x28,%esp
0x08048418 <+6>: and $0xfffffffff0,%esp
0x0804841f <+9>: mov $0xd,%eax
0x08048422 <+14>: add $0xf,%eax
0x08048425 <+17>: add $0x14,%eax
0x08048428 <+20>: shr $0x4,%eax
0x0804842b <+23>: sub $0x4,%eax
0x08048430 <+26>: movl $0x8048508,%esp
0x08048437 <+33>: call *0x8048500<printf@plt>
0x08048440 <+36>: movl $0x8048500,%eax
0x08048444 <+47>: call *0x8048500<printf@plt>
0x08048448 <+52>: lea -0x18(%ebp),%eax
0x0804844c <+55>: mov %eax,0x4(%esp)
0x0804844f <+59>: movl $0x804850c,%esp
0x08048452 <+62>: call *0x8048500<printf@plt>
0x08048456 <+71>: lea -0x18(%ebp),%eax
0x0804845a <+74>: mov $0x804850f,0x4(%esp)
=> 0x08048460 <+82>: mov %eax,%esp
0x08048463 <+85>: call *0x8048350<strcmp@plt>
0x08048466 <+92>: jne 0x8048477,<main+108>
0x08048471 <+95>: movl $0x8048509,%esp
0x08048474 <+101>: call *0x8048340<printf@plt>
0x08048477 <+106>: jmp 0x804844b,<main+120>
0x08048480 <+109>: movl $0x8048509,%esp
0x08048483 <+115>: call *0x8048350<strcmp@plt>
0x08048486 <+120>: mov $0x0,%eax
0x08048491 <+125>: leave
0x08048491 <+126>: ret
End of assembler dump.
(gdb) x /s $eax
0xfffffd100: "acbd"
(gdb) x /s 0x8048508
0xfffffd101: "250382"
(gdb) 
```

15. As the value **250382** is being compared to our input value **acbd**, this means that the **password should be 250382**
16. Close the debugger using **quit** and launch the binary normally.
17. Enter the password.
18. Validate the result.

Output



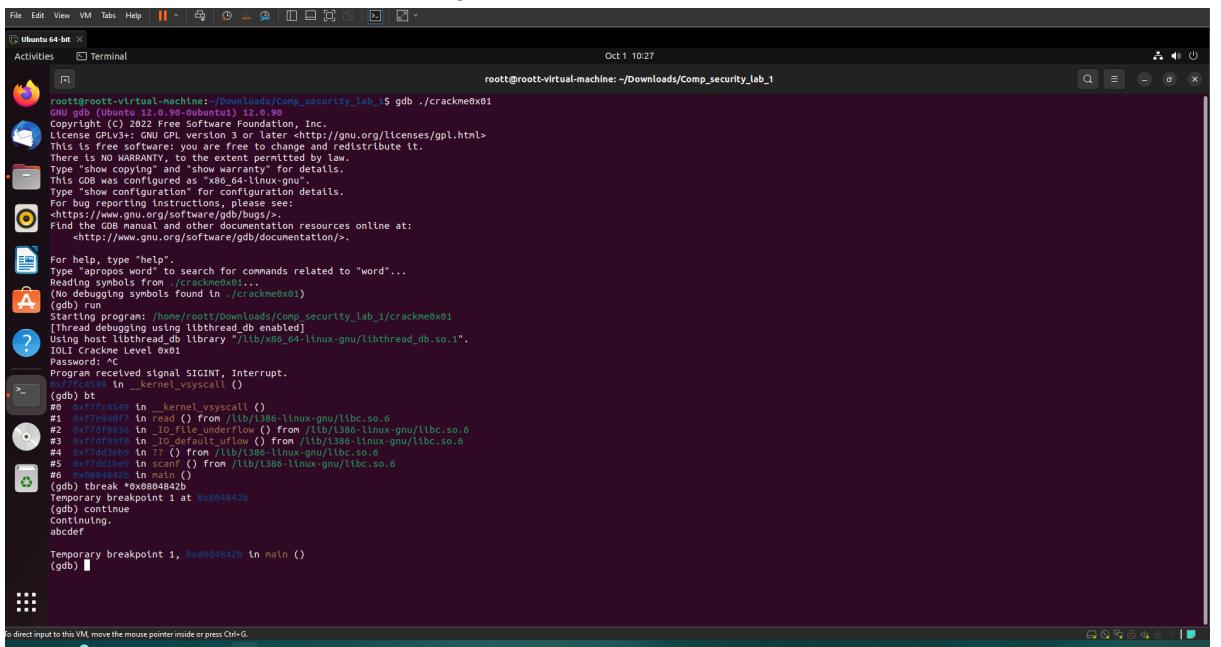
The screenshot shows a terminal window on an Ubuntu 64-bit system. The root user has run the program `./crackme0x00`. The output shows the password was entered correctly, and the user is now a root shell.

```
root@root-virtual-machine:/Downloads/Comp_security_lab_1$ ./crackme0x00
Input Cracking level:0x00
Password:250382
Password OK :)
root@root-virtual-machine:/Downloads/Comp_security_lab_1$ 
```

Binary 2 - Crackme0x01 - Password: 5274

Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **abcdef** and **press enter**
8. The screen should like like following



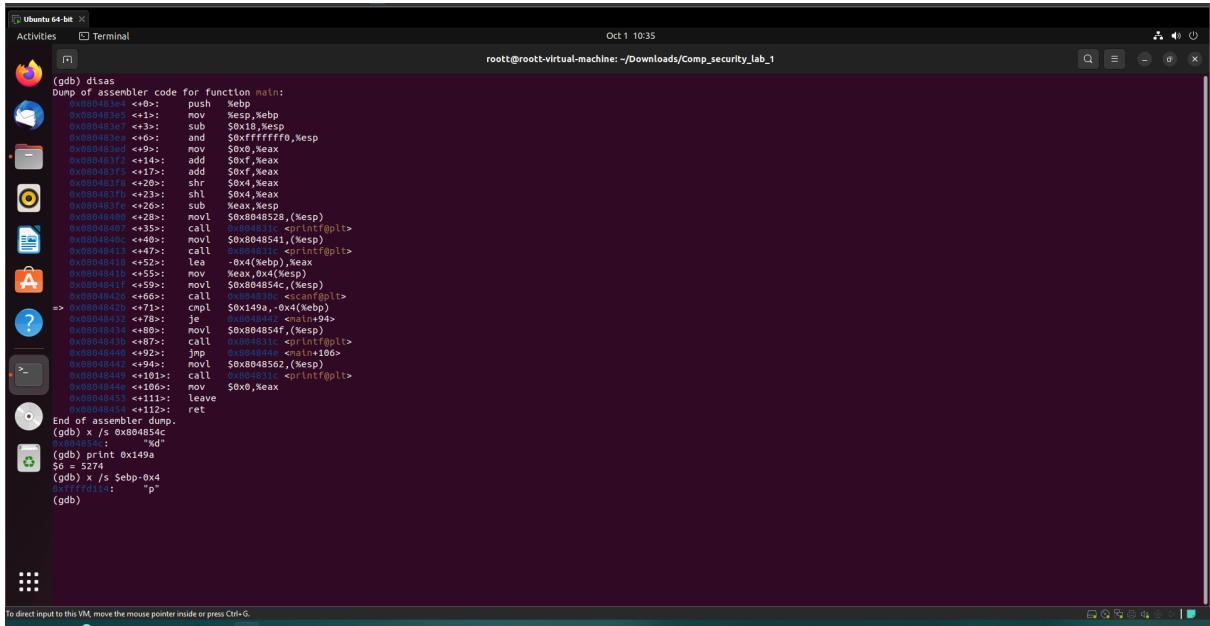
```
root@roott@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ gdb ./crackme0x01
GNU gdb (Ubuntu 12.1-90-Dubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
Type "show configuration" for configuration details.
Type "bug" 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"...
Reading symbols from ./crackme0x01...
(No debugging symbols found in ./crackme0x01)
(gdb) run
Starting program: /home/roott/Downloads/Comp_security_lab_1/crackme0x01
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
IOLI Crackme Level 0x01
Program received signal SIGINT, Interrupt.
0x7f7fc4549 in _kernel_vsyscall ()
(gdb) bt
#0 0x7f7fc4549 in _kernel_vsyscall ()
#1 0xf7d7ff836 in __IO_file_underflow () from /lib/i386-linux-gnu/libc.so.6
#2 0xf7d7ff990 in __IO_default_uflow () from /lib/i386-linux-gnu/libc.so.6
#3 0xf7d7d1be9 in scanf () from /lib/i386-linux-gnu/libc.so.6
#4 0xf7d7d1be9 in __isoc99_scanf () from /lib/i386-linux-gnu/libc.so.6
(gdb) tbreak *0x8004842b
Temporary breakpoint 1 at 0x8004842b
(gdb) continue
Continuing.
abcdef

Temporary breakpoint 1, 0x0004842b in main ()
(gdb) 
```

9. Disassemble the binary using **disas**
10. Find the input format for **scanf**
11. Find the comparison line **cmp**
12. Find the values being compared by looking into the individual values

13. The output should look like the following screen



The screenshot shows a terminal window titled "Ubuntu 64-bit" running on a Linux desktop environment. The command "dlsas" has been run in GDB, displaying the assembly code for the main function. The assembly code includes instructions for pushing and popping the stack, performing arithmetic operations like add and sub, and calling printf functions. The code ends with a "leave" instruction and a "ret" instruction. Below the assembly dump, there is a command history showing the execution of "x/s \$ebp-\$0x4", "print \$0", and "quit". The terminal window also displays the date and time as Oct 1 10:35.

```
(gdb) dlsas
Dump of assembler code for function main:
0x0004033c <+0>: push %ebp
0x0004033e <+1>: mov %esp,%ebp
0x0004033f <+2>: sub $0x18,%esp
0x00040340 <+3>: and $0xfffffff0,%esp
0x00040341 <+4>: mov %esp,%ebp
0x00040372 <+14>: add $0xf,%eax
0x00040375 <+17>: add $0xf,%eax
0x0004037f <+20>: shr $0x4,%eax
0x00040381 <+23>: sub $0x4,%eax
0x00040382 <+24>: movl $0x8048528,%esp
0x00040387 <+28>: movl $0x8048528,%esp
0x00040390 <+35>: call *0x804851c<>printf@plt>
0x00040393 <+40>: movl $0x8048541,%esp
0x00040395 <+47>: call *0x804851c<>printf@plt>
0x00040398 <+50>: lea -0x4(%ebp),%eax
0x0004039b <+53>: mov %eax,0x4(%esp)
0x000403fb <+59>: movl $0x804854c,%esp
0x00040426 <+66>: call *0x804830c<>scanf@plt>
=> 0x0004042b <+71>: cmpl $0x149a,-0x4(%ebp)
0x00040432 <+78>: je -0x4(%ebp),main+9d
0x00040435 <+81>: movl $0x804854f,-0x4(%ebp)
0x0004043b <+87>: call *0x804851c<>printf@plt>
0x00040444 <+92>: jmp $0x0004044e<>main+106>
0x00040442 <+94>: movl $0x8048562,%esp
0x00040449 <+101>: call *0x804851c<>printf@plt>
0x00040453 <+106>: mov $0x0,%eax
0x00040453 <+111>: leave
0x00040454 <+112>: ret
End of assembler dump.
(gdb) x /s $ebp-$0x4
$0 = 5274
(gdb) x /s $ebp-$0x4
$0ffffd11: "p"
(gdb)
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

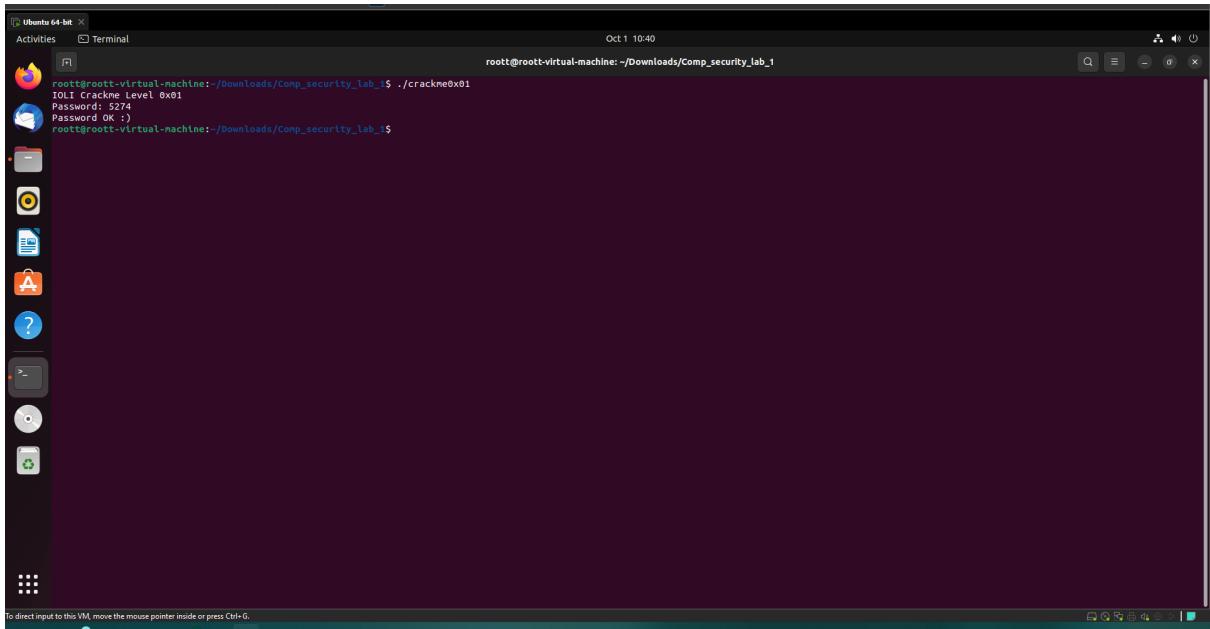
14. As the literal value of 0x149a in hex or 5274 in decimal is being compared to a variable, this means that the **password should be 5274**.

15. Close the debugger using **quit** and launch the binary normally.

16. Enter the password.

17. Validate the result.

Output



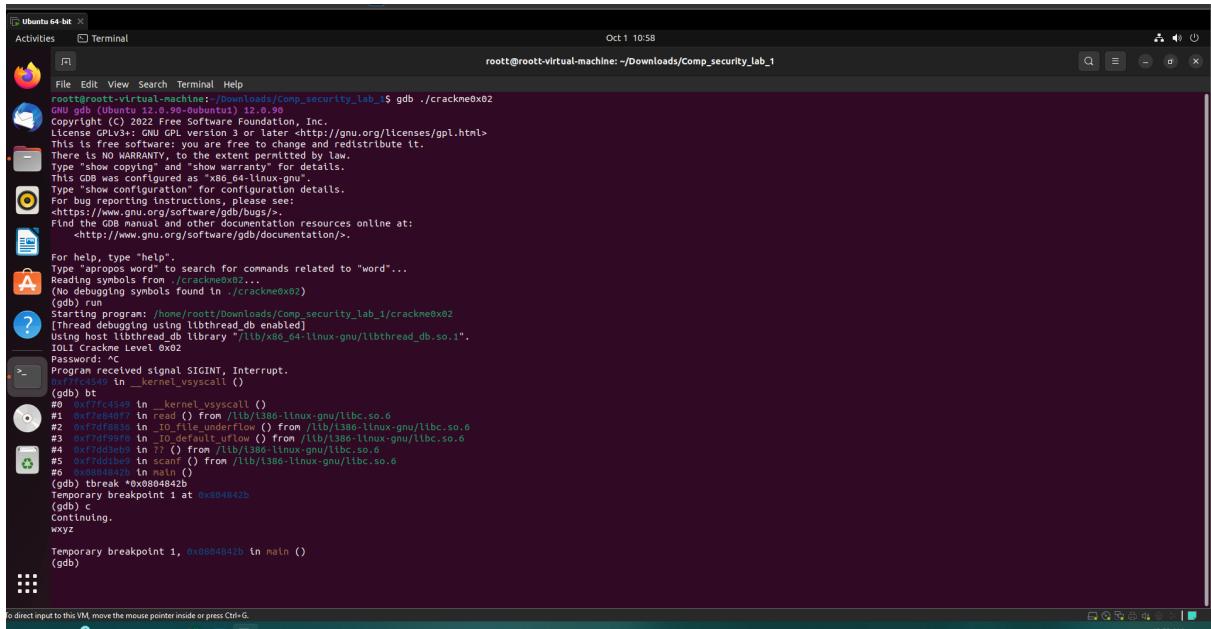
The screenshot shows a terminal window titled "Ubuntu 64-bit" running on a Linux desktop environment. The command "crackme0x01" has been run, and the output shows the password was entered correctly. The terminal window also displays the date and time as Oct 1 10:40.

```
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1$ ./crackme0x01
T0LT Crackme Level 0x01
Password: 5274
Password OK :)
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1$
```

Binary 3 - Crackme0x02 - Password: 338724

Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **998** and **press enter**
8. The screen should like like following

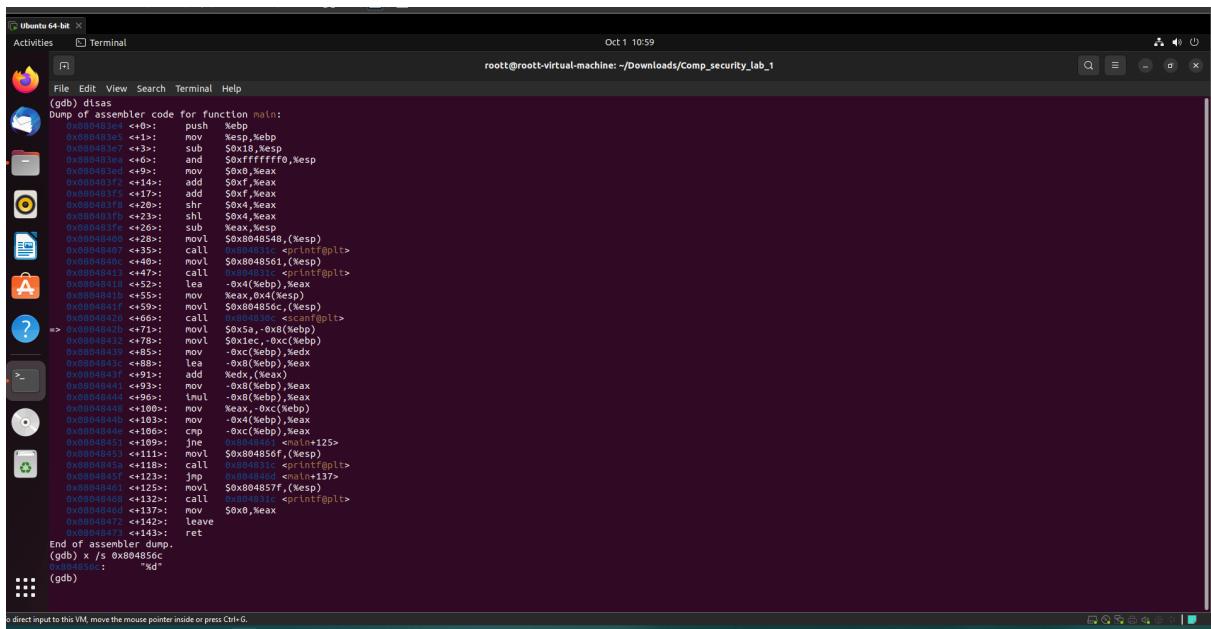


```
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
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"...
Reading symbols from /home/roott/Downloads/Comp_security_lab_1/crackme0x02...
(gdb) Starting program: /home/roott/Downloads/Comp_security_lab_1/crackme0x02
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
IOLI Crackme Level: 0x02
Password: "998"
Program received signal SIGINT, Interrupt.
#0 0xf7f4549 in __kernel_vsyscall ()
(gdb) bt
#0 0xf7f4549 in __kernel_vsyscall ()
#1 0xf7f8407 in read () from /lib/x86_64-linux-gnu/libc.so.6
#2 0xf7d4990 in __IO_default_uflow () from /lib/x86_64-linux-gnu/libc.so.6
#3 0xf7d4980 in scanf () from /lib/x86_64-linux-gnu/libc.so.6
#4 0xf7d3e9b in ?? () from /lib/x86_64-linux-gnu/libc.so.6
#5 0xf7d1b9 in scanf () from /lib/x86_64-linux-gnu/libc.so.6
#6 0x0004042b in main ()
(gdb) tbreak *0x0004042b
Temporary breakpoint 1 at 0x0004042b
(gdb) c
Continuing.
wxz
Temporary breakpoint 1, 0x0004042b in main ()
(gdb)
```

9. Disassemble the binary using **disas**

10. Find the input format for **scanf**



```
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
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".
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/>.

(gdb) disas
Dump of assembler code for function main:
0x000403e1 <+0>: push %ebp
0x000403e1 <+1>: mov %esp,%ebp
0x000403e3 <+3>: sub $0x1,%esp
0x000403e5 <+5>: add $0xffffffff,%esp
0x000403e7 <+7>: mov %eax,%eax
0x000403f2 <+14>: add %eax,%eax
0x000403f4 <+17>: add %eax,%eax
0x000403f6 <+20>: shr $0x4,%eax
0x000403f8 <+22>: sbb $0x1,%eax
0x000403fa <+26>: sub %eax,%esp
0x00040400 <+28>: movl $0x00405418,%esp
0x00040407 <+35>: call $0x004031,<printf@plt>
0x00040410 <+40>: mov $0x0040551,%esp
0x00040412 <+42>: call $0x004030,<scanf@plt>
0x00040414 <+45>: mov %eax,%esp
0x00040415 <+52>: lea %eax,[%esp]
0x0004041b <+55>: mov %eax,%esp
0x00040420 <+60>: call $0x004030,<scanf@plt>
0x00040422 <+62>: mov $0x00405459,%esp
0x00040423 <+65>: movl $0x00405459,%esp
0x00040425 <+70>: movl $0x00405459,%esp
0x00040430 <+85>: movl $0x00405459,%esp
0x00040431 <+88>: lea %edx,(%eax)
0x00040433 <+91>: add %edx,(%eax)
0x00040434 <+94>: movl $0x00405459,%esp
0x00040440 <+100>: mov %eax,-0x4(%ebp)
0x00040441 <+103>: movl $0x00405459,%esp
0x00040444 <+106>: cmp -0x4(%ebp),%eax
0x00040453 <+109>: jne $0x00405459,<scanf@plt>
0x00040455 <+111>: movl $0x00405459,%esp
0x00040457 <+118>: call $0x004030,<scanf@plt>
0x00040459 <+123>: jmp $0x0040461,<nain+137>
0x00040461 <+125>: movl $0x004057f,%esp
0x00040464 <+132>: call $0x004030,<scanf@plt>
0x00040466 <+134>: movl $0x00405459,%esp
0x00040471 <+142>: leave
0x00040473 <+143>: ret
End of assembler dump.
(gdb) x /s 0x0040551
0x0040551: "kd"
```

11. Add a breakpoint on the comparison line using **tbreak *<memory address of line of comparison>**

12. Continue code execution using **continue**

13. Disassemble the binary using **disas**

The screenshot shows a terminal window titled "Ubuntu 64-bit" with the command "root@root-virtual-machine: ~/Downloads/Comp_security_lab_1". The assembly code for the main function is displayed, showing various instructions like movl, call, and cmp. A temporary breakpoint is set at address 0x0804844e. The command "(gdb) disas" is used to list the assembly code. The assembly code includes instructions for printing strings, setting up memory, and performing comparisons.

14. Find the comparison line **cmp**

15. Find the values being compared by looking into the individual values

16. The output should look like the following screen

The screenshot shows a terminal window titled "Ubuntu 64-bit" with the command "root@root-virtual-machine: ~/Downloads/Comp_security_lab_1". The assembly code for the main function is displayed, showing various instructions like push, mov, and cmp. The register \$eax is printed with the value 998, and the register \$ebp is printed with the value 338724. The assembly code includes instructions for printing strings, setting up memory, and performing comparisons.

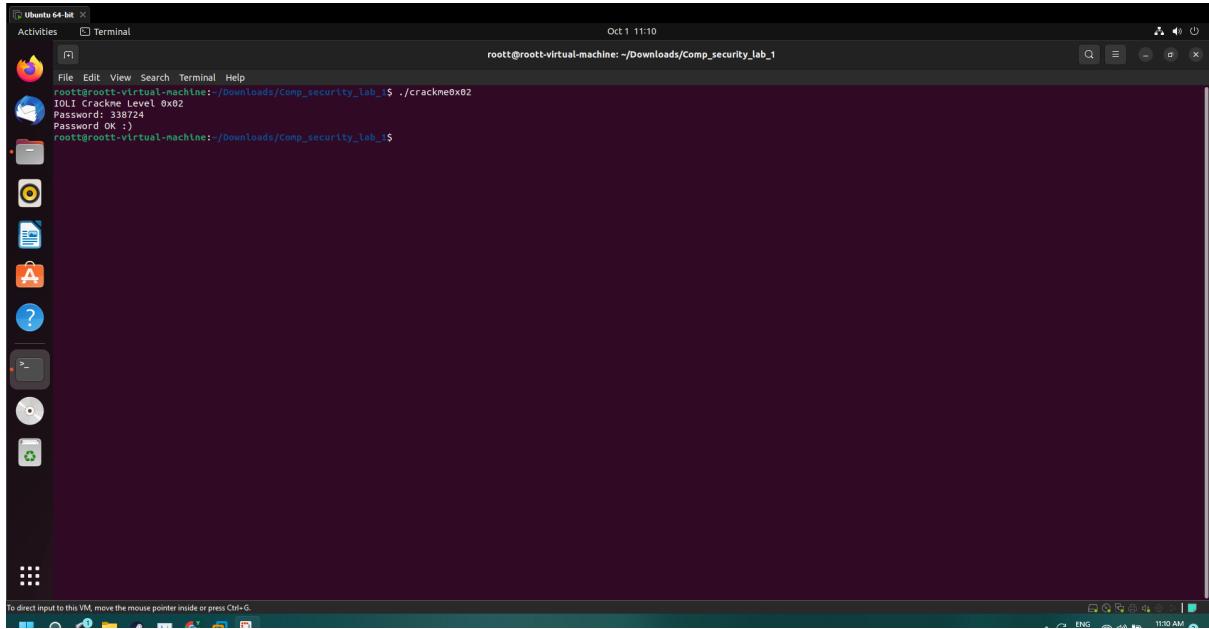
17. As the **value of register eax = 998** is being compared to the **value of \$ebp-0xc = 338724**, this means that **338724** is our password.

18. Close the debugger using **quit** and launch the binary normally.

19. Enter the password.

20. Validate the result.

Output



The screenshot shows a terminal window titled "Terminal" in an "Ubuntu 64-bit" desktop environment. The window title bar includes the date and time: "Oct 1 11:10". The terminal content displays the following text:

```
root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ ./crackme0x02
ID11 Crackme Level 0x02
Password: 338724
Password OK :)
```

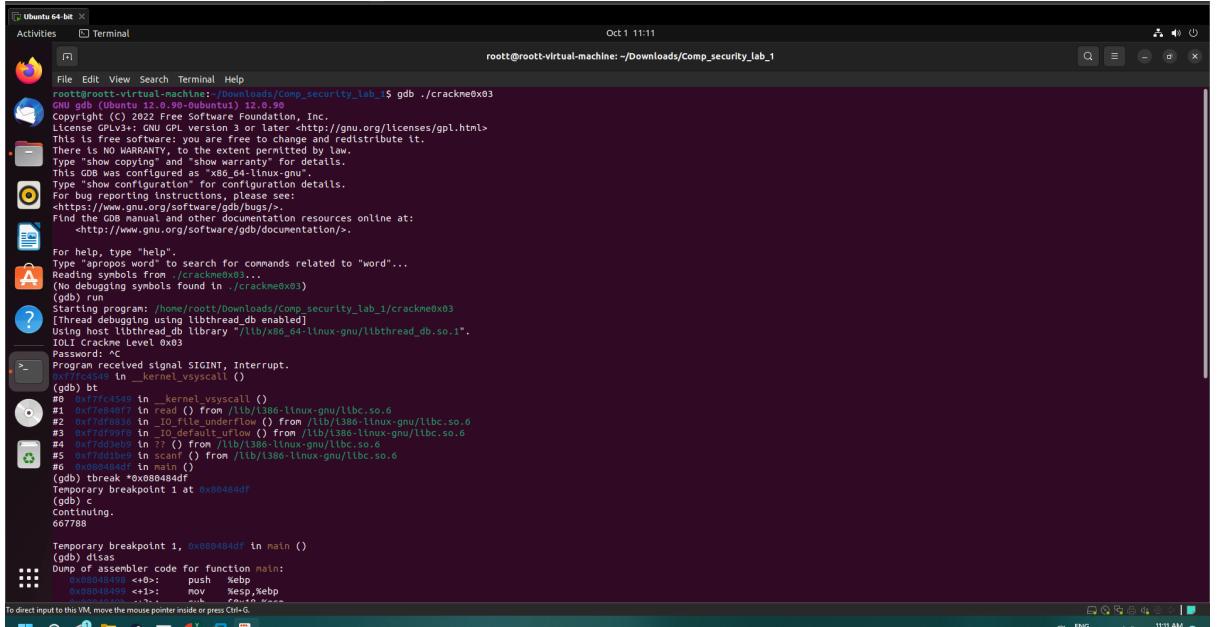
The terminal window has a dark background with light-colored text. The left sidebar contains icons for various applications like a browser, file manager, and system settings. The bottom status bar shows system information and the current time.

Binary 4 - Crackme0x03 - Password: 338724

Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **667788** and press enter

8. The screen should like like following



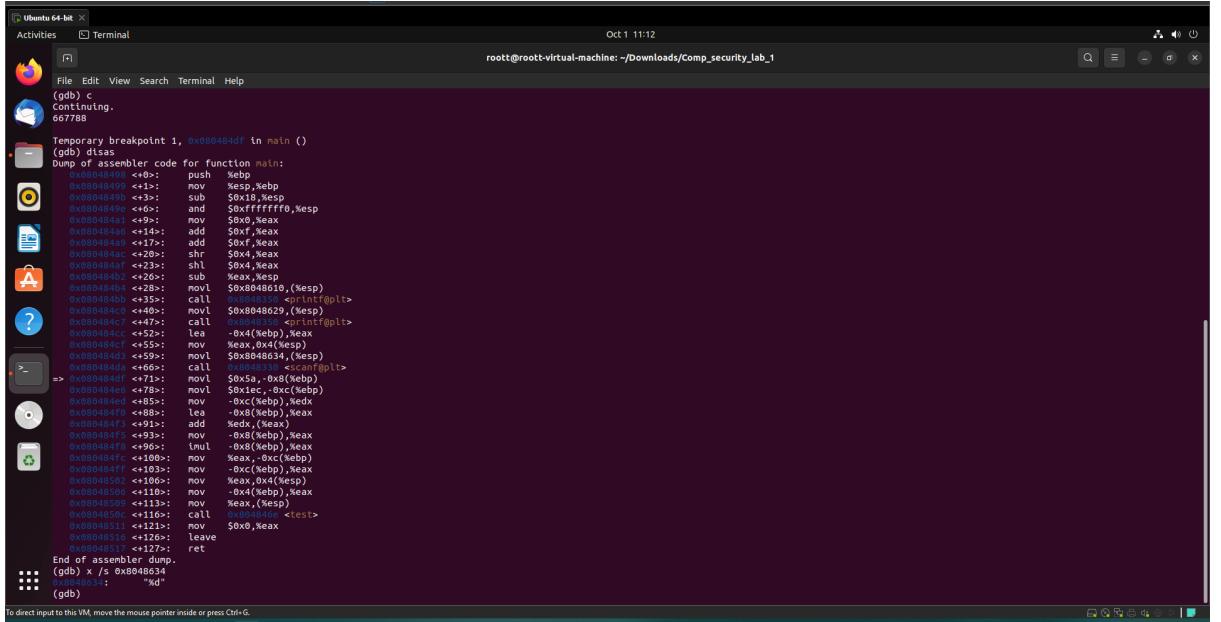
```
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1$ gdb ./crackme0x03
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show configuration" or "show build" for configuration 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:
<https://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Type "symbol-symbol" to show symbols from ./crackme0x03...
(No debugging symbols found in ./crackme0x03)
(gdb) run
Starting program: /home/root/Downloads/Comp_security_lab_1/crackme0x03
[Thread debugging using libthread_db enabled]
[New Thread 0x7f7fc540 in __kernel_vsyscall ()]
IOLI Crackme Level 0x03
Password: ~c
Program received signal SIGINT, Interrupt.
0x7f7fc549 in __kernel_vsyscall ()
(gdb) bt
#1 0x7f7e8407 in read () from /lib/x86_64-linux-gnu/libc.so.6
#2 0x7f7d80836 in __io_file_underflow () from /lib/x86_64-linux-gnu/libc.so.6
#3 0x7f7d990 in __io_default_uflow () from /lib/x86_64-linux-gnu/libc.so.6
#4 0x7f7d34b5 in __read_nocancel () from /lib/x86_64-linux-gnu/libc.so.6
#5 0x7f7d34b5 in __read_nocancel () from /lib/x86_64-linux-gnu/libc.so.6
#6 0x0000000 in main ()
(gdb) bbreak *0x000484df
Temporary breakpoint 1 at 0x80484df
(gdb) c
Continuing.
667788

Temporary breakpoint 1, 0x000484df in main ()
(gdb) disas
Dump of assembler code for function main:
0x00048409 <+0>:    push  %rbp
0x00048409 <+1>:    mov   %rsp,%rbp
0x00048410 <+2>:    sub   $0x18,%rbp
0x00048411 <+3>:    mov   $0x0,%eax
0x00048412 <+4>:    add   $0x10,%rbp
0x00048413 <+5>:    mov   $0x4,%eax
0x00048414 <+6>:    shr   $0x4,%eax
0x00048415 <+7>:    mov   $0x4,%eax
0x00048416 <+8>:    shr   $0x4,%eax
0x00048417 <+9>:    mov   $0x4,%eax
0x00048418 <+10>:   shr  $0x4,%eax
0x00048419 <+11>:   mov   $0x4,%eax
0x0004841a <+12>:   shr  $0x4,%eax
0x0004841b <+13>:   mov   $0x4,%eax
0x0004841c <+14>:   shr  $0x4,%eax
0x0004841d <+15>:   mov   $0x4,%eax
0x0004841e <+16>:   shr  $0x4,%eax
0x0004841f <+17>:   mov   $0x4,%eax
0x00048420 <+18>:   shr  $0x4,%eax
0x00048421 <+19>:   mov   $0x4,%eax
0x00048422 <+20>:   shr  $0x4,%eax
0x00048423 <+21>:   mov   $0x4,%eax
0x00048424 <+22>:   shr  $0x4,%eax
0x00048425 <+23>:   mov   $0x4,%eax
0x00048426 <+24>:   shr  $0x4,%eax
0x00048427 <+25>:   mov   $0x4,%eax
0x00048428 <+26>:   shr  $0x4,%eax
0x00048429 <+27>:   mov   $0x4,%eax
0x0004842a <+28>:   shr  $0x4,%eax
0x0004842b <+29>:   mov   $0x4,%eax
0x0004842c <+30>:   shr  $0x4,%eax
0x0004842d <+31>:   mov   $0x4,%eax
0x0004842e <+32>:   shr  $0x4,%eax
0x0004842f <+33>:   mov   $0x4,%eax
0x00048430 <+34>:   shr  $0x4,%eax
0x00048431 <+35>:   mov   $0x4,%eax
0x00048432 <+36>:   shr  $0x4,%eax
0x00048433 <+37>:   mov   $0x4,%eax
0x00048434 <+38>:   shr  $0x4,%eax
0x00048435 <+39>:   mov   $0x4,%eax
0x00048436 <+40>:   shr  $0x4,%eax
0x00048437 <+41>:   mov   $0x4,%eax
0x00048438 <+42>:   shr  $0x4,%eax
0x00048439 <+43>:   mov   $0x4,%eax
0x0004843a <+44>:   shr  $0x4,%eax
0x0004843b <+45>:   mov   $0x4,%eax
0x0004843c <+46>:   shr  $0x4,%eax
0x0004843d <+47>:   mov   $0x4,%eax
0x0004843e <+48>:   shr  $0x4,%eax
0x0004843f <+49>:   mov   $0x4,%eax
0x00048440 <+50>:   shr  $0x4,%eax
0x00048441 <+51>:   mov   $0x4,%eax
0x00048442 <+52>:   shr  $0x4,%eax
0x00048443 <+53>:   mov   $0x4,%eax
0x00048444 <+54>:   shr  $0x4,%eax
0x00048445 <+55>:   mov   $0x4,%eax
0x00048446 <+56>:   shr  $0x4,%eax
0x00048447 <+57>:   mov   $0x4,%eax
0x00048448 <+58>:   shr  $0x4,%eax
0x00048449 <+59>:   mov   $0x4,%eax
0x0004844a <+60>:   shr  $0x4,%eax
0x0004844b <+61>:   mov   $0x4,%eax
0x0004844c <+62>:   shr  $0x4,%eax
0x0004844d <+63>:   mov   $0x4,%eax
0x0004844e <+64>:   shr  $0x4,%eax
0x0004844f <+65>:   mov   $0x4,%eax
0x00048450 <+66>:   shr  $0x4,%eax
0x00048451 <+67>:   mov   $0x4,%eax
0x00048452 <+68>:   shr  $0x4,%eax
0x00048453 <+69>:   mov   $0x4,%eax
0x00048454 <+70>:   shr  $0x4,%eax
0x00048455 <+71>:   mov   $0x4,%eax
0x00048456 <+72>:   shr  $0x4,%eax
0x00048457 <+73>:   mov   $0x4,%eax
0x00048458 <+74>:   shr  $0x4,%eax
0x00048459 <+75>:   mov   $0x4,%eax
0x0004845a <+76>:   shr  $0x4,%eax
0x0004845b <+77>:   mov   $0x4,%eax
0x0004845c <+78>:   shr  $0x4,%eax
0x0004845d <+79>:   mov   $0x4,%eax
0x0004845e <+80>:   shr  $0x4,%eax
0x0004845f <+81>:   mov   $0x4,%eax
0x00048460 <+82>:   shr  $0x4,%eax
0x00048461 <+83>:   mov   $0x4,%eax
0x00048462 <+84>:   shr  $0x4,%eax
0x00048463 <+85>:   mov   $0x4,%eax
0x00048464 <+86>:   shr  $0x4,%eax
0x00048465 <+87>:   mov   $0x4,%eax
0x00048466 <+88>:   shr  $0x4,%eax
0x00048467 <+89>:   mov   $0x4,%eax
0x00048468 <+90>:   shr  $0x4,%eax
0x00048469 <+91>:   mov   $0x4,%eax
0x0004846a <+92>:   shr  $0x4,%eax
0x0004846b <+93>:   mov   $0x4,%eax
0x0004846c <+94>:   shr  $0x4,%eax
0x0004846d <+95>:   mov   $0x4,%eax
0x0004846e <+96>:   shr  $0x4,%eax
0x0004846f <+97>:   mov   $0x4,%eax
0x00048470 <+98>:   shr  $0x4,%eax
0x00048471 <+99>:   mov   $0x4,%eax
0x00048472 <+100>:  shr  $0x4,%eax
0x00048473 <+101>:  mov   $0x4,%eax
0x00048474 <+102>:  shr  $0x4,%eax
0x00048475 <+103>:  mov   $0x4,%eax
0x00048476 <+104>:  shr  $0x4,%eax
0x00048477 <+105>:  mov   $0x4,%eax
0x00048478 <+106>:  shr  $0x4,%eax
0x00048479 <+107>:  mov   $0x4,%eax
0x0004847a <+108>:  shr  $0x4,%eax
0x0004847b <+109>:  mov   $0x4,%eax
0x0004847c <+110>:  shr  $0x4,%eax
0x0004847d <+111>:  mov   $0x4,%eax
0x0004847e <+112>:  shr  $0x4,%eax
0x0004847f <+113>:  mov   $0x4,%eax
0x00048480 <+114>:  shr  $0x4,%eax
0x00048481 <+115>:  mov   $0x4,%eax
0x00048482 <+116>:  shr  $0x4,%eax
0x00048483 <+117>:  leave
0x00048484 <+118>:  ret
End of assembler dump.
```

9. Disassemble the binary using **disas**

10. Find the input format for **scanf**



```
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1$ gdb ./crackme0x03
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 configuration" or "show build" for configuration 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:
<https://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Type "symbol-symbol" to show symbols from ./crackme0x03...
(No debugging symbols found in ./crackme0x03)
(gdb) c
Continuing.
667788

Temporary breakpoint 1, 0x000484df in main ()
(gdb) disas
Dump of assembler code for function main:
0x00048409 <+0>:    push  %rbp
0x00048409 <+1>:    mov   %rsp,%rbp
0x00048410 <+2>:    sub   $0x18,%rbp
0x00048411 <+3>:    mov   $0x0,%eax
0x00048412 <+4>:    add   $0x10,%rbp
0x00048413 <+5>:    mov   $0x4,%eax
0x00048414 <+6>:    shr  $0x4,%eax
0x00048415 <+7>:    mov   $0x4,%eax
0x00048416 <+8>:    shr  $0x4,%eax
0x00048417 <+9>:    mov   $0x4,%eax
0x00048418 <+10>:   shr  $0x4,%eax
0x00048419 <+11>:   mov   $0x4,%eax
0x0004841a <+12>:   shr  $0x4,%eax
0x0004841b <+13>:   mov   $0x4,%eax
0x0004841c <+14>:   shr  $0x4,%eax
0x0004841d <+15>:   mov   $0x4,%eax
0x0004841e <+16>:   shr  $0x4,%eax
0x0004841f <+17>:   mov   $0x4,%eax
0x00048420 <+18>:   shr  $0x4,%eax
0x00048421 <+19>:   mov   $0x4,%eax
0x00048422 <+20>:   shr  $0x4,%eax
0x00048423 <+21>:   mov   $0x4,%eax
0x00048424 <+22>:   shr  $0x4,%eax
0x00048425 <+23>:   mov   $0x4,%eax
0x00048426 <+24>:   shr  $0x4,%eax
0x00048427 <+25>:   mov   $0x4,%eax
0x00048428 <+26>:   shr  $0x4,%eax
0x00048429 <+27>:   mov   $0x4,%eax
0x0004842a <+28>:   shr  $0x4,%eax
0x0004842b <+29>:   mov   $0x4,%eax
0x0004842c <+30>:   shr  $0x4,%eax
0x0004842d <+31>:   mov   $0x4,%eax
0x0004842e <+32>:   shr  $0x4,%eax
0x0004842f <+33>:   mov   $0x4,%eax
0x00048430 <+34>:   shr  $0x4,%eax
0x00048431 <+35>:   mov   $0x4,%eax
0x00048432 <+36>:   shr  $0x4,%eax
0x00048433 <+37>:   mov   $0x4,%eax
0x00048434 <+38>:   shr  $0x4,%eax
0x00048435 <+39>:   mov   $0x4,%eax
0x00048436 <+40>:   shr  $0x4,%eax
0x00048437 <+41>:   mov   $0x4,%eax
0x00048438 <+42>:   shr  $0x4,%eax
0x00048439 <+43>:   mov   $0x4,%eax
0x0004843a <+44>:   shr  $0x4,%eax
0x0004843b <+45>:   mov   $0x4,%eax
0x0004843c <+46>:   shr  $0x4,%eax
0x0004843d <+47>:   mov   $0x4,%eax
0x0004843e <+48>:   shr  $0x4,%eax
0x0004843f <+49>:   mov   $0x4,%eax
0x00048440 <+50>:   shr  $0x4,%eax
0x00048441 <+51>:   mov   $0x4,%eax
0x00048442 <+52>:   shr  $0x4,%eax
0x00048443 <+53>:   mov   $0x4,%eax
0x00048444 <+54>:   shr  $0x4,%eax
0x00048445 <+55>:   mov   $0x4,%eax
0x00048446 <+56>:   shr  $0x4,%eax
0x00048447 <+57>:   mov   $0x4,%eax
0x00048448 <+58>:   shr  $0x4,%eax
0x00048449 <+59>:   mov   $0x4,%eax
0x0004844a <+60>:   shr  $0x4,%eax
0x0004844b <+61>:   mov   $0x4,%eax
0x0004844c <+62>:   shr  $0x4,%eax
0x0004844d <+63>:   mov   $0x4,%eax
0x0004844e <+64>:   shr  $0x4,%eax
0x0004844f <+65>:   mov   $0x4,%eax
0x00048450 <+66>:   shr  $0x4,%eax
0x00048451 <+67>:   mov   $0x4,%eax
0x00048452 <+68>:   shr  $0x4,%eax
0x00048453 <+69>:   mov   $0x4,%eax
0x00048454 <+70>:   shr  $0x4,%eax
0x00048455 <+71>:   mov   $0x4,%eax
0x00048456 <+72>:   shr  $0x4,%eax
0x00048457 <+73>:   mov   $0x4,%eax
0x00048458 <+74>:   shr  $0x4,%eax
0x00048459 <+75>:   mov   $0x4,%eax
0x0004845a <+76>:   shr  $0x4,%eax
0x0004845b <+77>:   mov   $0x4,%eax
0x0004845c <+78>:   shr  $0x4,%eax
0x0004845d <+79>:   mov   $0x4,%eax
0x0004845e <+80>:   shr  $0x4,%eax
0x0004845f <+81>:   mov   $0x4,%eax
0x00048460 <+82>:   shr  $0x4,%eax
0x00048461 <+83>:   mov   $0x4,%eax
0x00048462 <+84>:   shr  $0x4,%eax
0x00048463 <+85>:   mov   $0x4,%eax
0x00048464 <+86>:   shr  $0x4,%eax
0x00048465 <+87>:   mov   $0x4,%eax
0x00048466 <+88>:   shr  $0x4,%eax
0x00048467 <+89>:   mov   $0x4,%eax
0x00048468 <+90>:   shr  $0x4,%eax
0x00048469 <+91>:   mov   $0x4,%eax
0x0004846a <+92>:   shr  $0x4,%eax
0x0004846b <+93>:   mov   $0x4,%eax
0x0004846c <+94>:   shr  $0x4,%eax
0x0004846d <+95>:   mov   $0x4,%eax
0x0004846e <+96>:   shr  $0x4,%eax
0x0004846f <+97>:   mov   $0x4,%eax
0x00048470 <+98>:   shr  $0x4,%eax
0x00048471 <+99>:   mov   $0x4,%eax
0x00048472 <+100>:  shr  $0x4,%eax
0x00048473 <+101>:  mov   $0x4,%eax
0x00048474 <+102>:  shr  $0x4,%eax
0x00048475 <+103>:  mov   $0x4,%eax
0x00048476 <+104>:  shr  $0x4,%eax
0x00048477 <+105>:  mov   $0x4,%eax
0x00048478 <+106>:  shr  $0x4,%eax
0x00048479 <+107>:  mov   $0x4,%eax
0x0004847a <+108>:  shr  $0x4,%eax
0x0004847b <+109>:  mov   $0x4,%eax
0x0004847c <+110>:  shr  $0x4,%eax
0x0004847d <+111>:  mov   $0x4,%eax
0x0004847e <+112>:  shr  $0x4,%eax
0x0004847f <+113>:  mov   $0x4,%eax
0x00048480 <+114>:  shr  $0x4,%eax
0x00048481 <+115>:  mov   $0x4,%eax
0x00048482 <+116>:  shr  $0x4,%eax
0x00048483 <+117>:  leave
0x00048484 <+118>:  ret
End of assembler dump.
```

11. The comparison is not in this function but there is a function call made for the function **test**. We will be adding a debugger for the function **test** using **tbreak test** and then continue execution using **continue** and then disassemble the code again using **disas**.

Ubuntu 64-bit Terminal

```
File Edit View Search Terminal Help
0x000448516 <+126>: leave
0x000448517 <+127>: ret
End of assembler dump.
(gdb) x /s 0x000448514
0x000448514: 00
(gdb) tbreak test
Temporary breakpoint 1 at 0x000448474
(gdb) disas
Dump of assembler code for function main:
0x000448474 <+1>: push %rbp
0x000448475 <+2>: mov %rsp,%rbp
0x000448476 <+3>: sub $0x18,%esp
0x000448477 <+4>: and $0xfffffffff0,%esp
0x000448478 <+5>: mov $0x0,%eax
0x000448479 <+6>: movl %eax,-0x10(%rbp)
0x00044847a <+7>: add $0xf,%esp
0x00044847b <+8>: shr $0x4,%eax
0x00044847c <+9>: shl $0x4,%eax
0x00044847d <+10>: sub %eax,%esp
0x00044847e <+11>: movl %eax,-0x10(%rbp)
0x00044847f <+12>: call *%rip@plt
0x000448480 <+13>: movl $0x000448629,%esp
0x000448481 <+14>: movl $0x000448629,%esp
0x000448482 <+15>: call *%rip@plt
0x000448483 <+16>: lea -0x4(%rbp),%eax
0x000448484 <+17>: mov %eax,0x4(%esp)
0x000448485 <+18>: movl %esp,-0x10(%rbp)
0x000448486 <+19>: call *%rip@plt
=> 0x000448487 <+20>: movl $0x5a,-0x8(%rbp)
0x000448488 <+21>: movl $0x1ec,-0xc(%rbp)
0x000448489 <+22>: movl $0xc(%rbp),%edx
0x00044848a <+23>: lea -0x4(%rbp),%eax
0x00044848b <+24>: add %edx,%eax
0x00044848c <+25>: mov -0x8(%rbp),%eax
0x00044848d <+26>: imul -0x8(%rbp),%eax
0x00044848e <+27>: mov %eax,-0xc(%rbp)
0x00044848f <+28>: mov %eax,-0x4(%rbp)
0x000448490 <+29>: mov %eax,-0x4(%rbp)
0x000448491 <+30>: mov -0x4(%rbp),%eax
0x000448492 <+31>: mov %eax,(%esp)
0x000448493 <+32>: call *%rip@plt
0x000448494 <+33>: mov $0x0,%eax
0x000448495 <+34>: leave
0x000448496 <+35>: ret
0x000448497 <+36>: End of assembler dump.
```

(gdb)

To direct input to this VM, move the mouse pointer inside or press Ctrl-I.

```
Ubuntu 64-bit
Activities Terminal Oct 1 11:17
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1

File Edit View Search Terminal Help
0x000444db <+15>: call 0x0004431a <printf@plt>
0x000444e7 <+16>: movl $0x0048c29, (%ebp)
0x000444e7 <+17>: call 0x0004444f <printf@plt>
0x000444cc <+22>: lea -0x4(%ebp),%eax
0x000444cf <+25>: mov %eax,0x4(%esp)
0x000444d3 <+29>: movl $0x0048634, (%esp)
0x000444d3 <+30>: call 0x0004444f <printf@plt>
=> 0x000444df <+71>: movl $0x5a,-0x8(%ebp)
0x000444e6 <+78>: movl $0x1ec,-0x4(%ebp)
0x000444ed <+85>: movl -0x8(%ebp),%edx
0x000444f0 <+88>: lea -0x8(%ebp),%eax
0x000444f1 <+91>: add %eax,%eax
0x000444f5 <+93>: movl $0x1ec,-0x4(%ebp)
0x000444f8 <+96>: imul -0x8(%ebp),%eax
0x000444f8 <+100>: movl %eax,-0xc(%ebp)
0x000444f7 <+103>: movl -0xc(%ebp),%eax
0x00044500 <+106>: movl %eax,0x4(%esp)
0x00044500 <+107>: movl -0x4(%ebp),%eax
0x00044509 <+113>: movl %eax,0x4(%esp)
0x0004450c <+116>: call 0x0004444f <test>
0x00044511 <+121>: movl $0x0,%eax
0x00044516 <+126>: leave
0x00044516 <+127>: ret
End of assembler dump.
(gdb) c
Continuing.

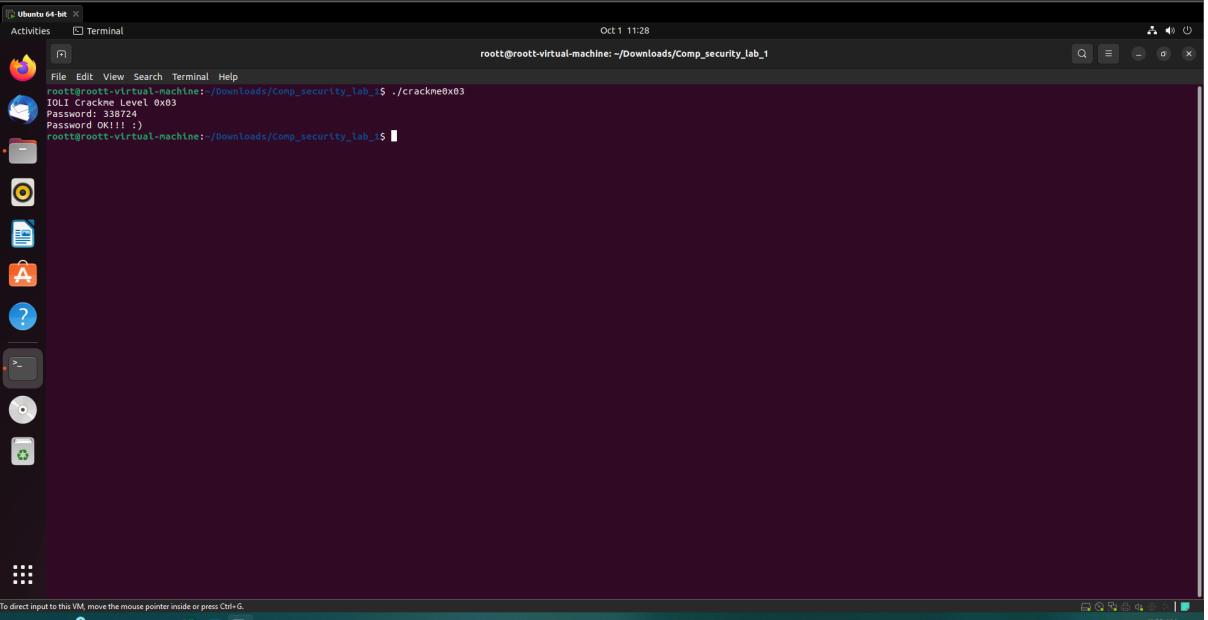
Temporary breakpoint 2, 0x0004474 in test ()
(gdb) disas
Dump of assembler code for function test:
0x0004446e <+0>: push %ebp
0x0004446f <+1>: mov %esp,%ebp
0x00044470 <+2>: sub $0x4,%ebp
=> 0x00044474 <+3>: movl $0x8(%ebp),%eax
0x00044477 <+9>: cmpq $0xc(%ebp),%eax
0x0004447a <+12>: je 0x0004448b <test+28>
0x0004447c <+14>: movl $0x80485ec, (%esp)
0x00044480 <+15>: call 0x00044411 <shl>
0x00044484 <+16>: jne 0x0004448b <test+48>
0x00044488 <+28>: movl $0x80485fe, (%esp)
0x00044491 <+35>: call 0x00044411 <shl>
0x00044494 <+40>: leave
0x00044494 <+41>: ret
End of assembler dump.
(gdb) 
```

```
Ubuntu 64-bit
Activities Terminal Oct 1 11:25
root@root-virtual-machine: ~/Downloads/Comp_security_lab_1

File Edit View Search Terminal Help
(gdb) disas
Dump of assembler code for function test:
0x0004446e <+0>: push %ebp
0x0004446f <+1>: mov %esp,%ebp
0x00044470 <+2>: sub $0x4,%ebp
=> 0x00044474 <+3>: movl $0x8(%ebp),%eax
0x00044477 <+9>: cmpq $0xc(%ebp),%eax
0x0004447a <+12>: je 0x0004448b <test+28>
0x00044480 <+15>: movl $0x80485ec, (%esp)
0x00044484 <+16>: call 0x00044411 <shl>
0x00044488 <+26>: jmp 0x00044494 <test+40>
0x0004448b <+28>: movl $0x80485fe, (%esp)
0x00044491 <+35>: call 0x00044411 <shl>
0x00044494 <+40>: leave
0x00044494 <+41>: ret
End of assembler dump.
(gdb) prime_seax
$1 = 667788
(gdb) x /d $ebp+0xc
$1 = 338724
(gdb) 
```

12. As the value of register eax = 667788 is being compared to the value of \$ebp+0xc = 338724, this means that 338724 is our password.
13. Close the debugger using quit and launch the binary normally.
14. Enter the password.
15. Validate the result.

Output



The screenshot shows a terminal window titled "Terminal" in an "Activities" dock. The window title bar includes the text "Ubuntu 64-bit", "Activities", "Terminal", and the date/time "Oct 1 11:28". The terminal content shows the following text:

```
root@roott-virtual-machine: ~/Downloads/Comp_security_lab_1$ ./crackme0x03
IOLT Crackme Level 0x03
Password: 338724
Password OK!!! :)
```

The terminal window has a dark background with light-colored text. The dock on the left contains icons for various applications like a browser, file manager, and system tools. The bottom of the screen shows the Unity interface with a green progress bar.

Binary 5 - Crackme0x04 - Password: Any number password with sum of digits = 15. For ex 12345 or 555 or 3336

Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
2. Run the binary using **run**
3. Interrupt using **ctrl+c**
4. Get the backtrace of code using **bt**
5. Add a breakpoint on main using **tbreak <memory address of main>**
6. Continue the code execution using **continue**
7. Enter a password - **32645** and press enter

8. The screen should like like following

```
root@root@root-virtual-machine: ~/Downloads/Comp_security_lab_1$ gdb ./crackme0x04
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2020 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 "i386-linux-gnu".
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./crackme0x04...
(No debugging symbols found in ./crackme0x04)

Starting program: /home/root/Downloads/Comp_security_lab_1/crackme0x04
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
IOLI Crackme Level 0x04
Password: ``

Program received signal SIGINT, Interrupt.
#0 0xf7fc5c49 in __kernel_vsyscall ()
(gdb) b
#0 0xf7fc5c49 in __kernel_vsyscall ()
#1 0x7e840f7 in read () from /lib/i386-linux-gnu/libc.so.6
#2 0x7d7f836 in __IO_file_underflow () from /lib/i386-linux-gnu/libc.so.6
#3 0x7d7f830 in __IO_default_uflow () from /lib/i386-linux-gnu/libc.so.6
#4 0xf7d3eb9 in ?? () from /lib/i386-linux-gnu/libc.so.6
#5 0xf7d1be9 in scanf () from /lib/i386-linux-gnu/libc.so.6
#6 0x00048553 in main ()
(gdb) bbreak *0x00048553
Temporary breakpoint 1 at 0x00048553
(gdb) c
Continuing.
32645

Temporary breakpoint 1, 0x00048553 in main ()
(gdb) disas
Dump of assembler code for function main:
0x00048559 <0x00048559>: push %ebp
0x0004855a <0x0004855a>: mov %esp,%ebp
0x0004855b <0x0004855b>: sub $0x8,%esp
0x0004855c <0x0004855c>: and %eax,%esp
0x0004855d <0x0004855d>: mov %eax,%esp
0x0004855e <0x0004855e>: mov %eax,%esp
0x0004855f <0x0004855f>: mov %eax,%esp
0x00048560 <0x00048560>: mov %eax,%esp
0x00048561 <0x00048561>: add %eax,%esp
0x00048562 <0x00048562>: add %eax,%esp
0x00048563 <0x00048563>: shr %eax,%esp
0x00048564 <0x00048564>: shr %eax,%esp
0x00048565 <0x00048565>: sub %eax,%esp
0x00048566 <0x00048566>: movl $0x804865e,%esp
0x00048567 <0x00048567>: call 0x00048534 <printf@plt>
0x00048568 <0x00048568>: movl $0x8048677,%esp
0x00048569 <0x00048569>: call 0x00048534 <printf@plt>
0x00048570 <0x00048570>: lea -0x78(%ebp),%eax
0x00048571 <0x00048571>: mov %eax,0x4(%esp)
0x00048572 <0x00048572>: movl $0x8048682,%esp
0x00048573 <0x00048573>: call 0x00048534 <scanf@plt>
0x00048574 <0x00048574>: lea -0x78(%ebp),%eax
0x00048575 <0x00048575>: mov %eax,0x4(%esp)
0x00048576 <0x00048576>: mov %eax,%esp
0x00048577 <0x00048577>: call 0x00048534 <check>
0x00048578 <0x00048578>: mov $0x0,%eax
0x00048579 <0x00048579>: leave
0x0004857a <0x0004857a>: ret
End of assembler dump.
(gdb) x /s 0x00048682
0x00048682: "%s"
(gdb) 
```

9. Disassemble the binary using disas

10. Find the input format for scanf

```
root@root@root-virtual-machine: ~/Downloads/Comp_security_lab_1$ gdb ./crackme0x04
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2020 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 "i386-linux-gnu".
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./crackme0x04...
(No debugging symbols found in ./crackme0x04)

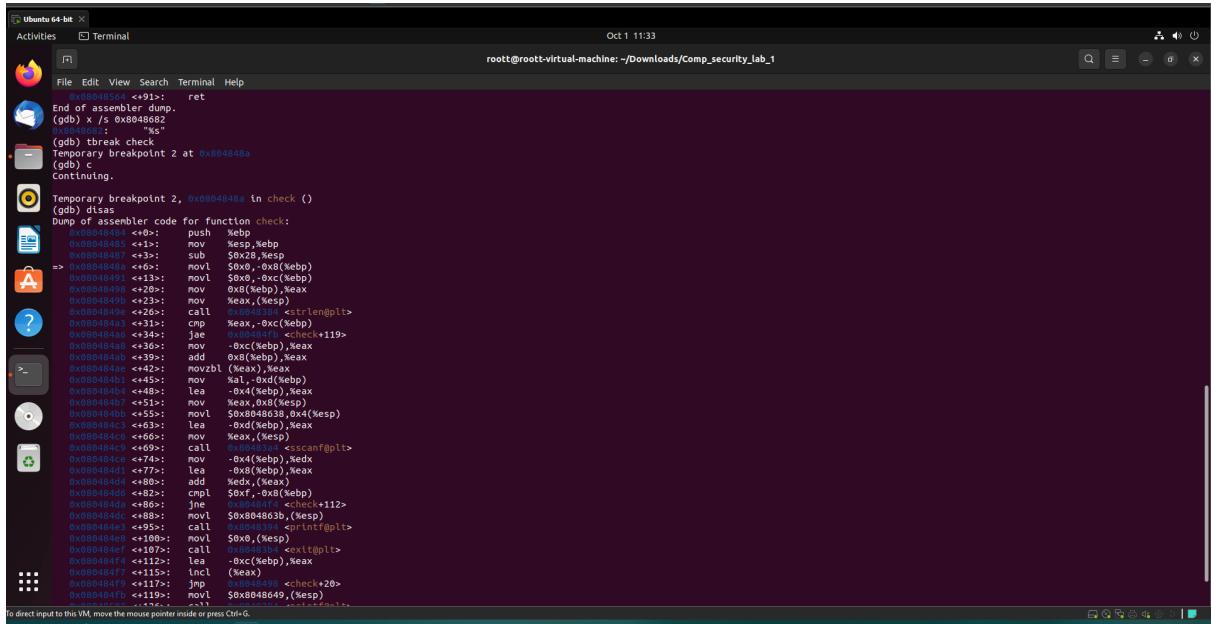
Starting program: /home/root/Downloads/Comp_security_lab_1/crackme0x04
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
IOLI Crackme Level 0x04
Password: ``

Program received signal SIGINT, Interrupt.
#0 0xf7fc5c49 in __kernel_vsyscall ()
#1 0x7e840f7 in read () from /lib/i386-linux-gnu/libc.so.6
#2 0x7d7f836 in __IO_file_underflow () from /lib/i386-linux-gnu/libc.so.6
#3 0x7d7f830 in __IO_default_uflow () from /lib/i386-linux-gnu/libc.so.6
#4 0xf7d3eb9 in ?? () from /lib/i386-linux-gnu/libc.so.6
#5 0xf7d1be9 in scanf () from /lib/i386-linux-gnu/libc.so.6
#6 0x00048553 in main ()
(gdb) bbreak *0x00048553
Temporary breakpoint 1 at 0x00048553
(gdb) c
Continuing.
32645

Temporary breakpoint 1, 0x00048553 in main ()
(gdb) disas
Dump of assembler code for function main:
0x00048559 <0x00048559>: push %ebp
0x0004855a <0x0004855a>: mov %esp,%ebp
0x0004855b <0x0004855b>: sub $0x8,%esp
0x0004855c <0x0004855c>: and %eax,%esp
0x0004855d <0x0004855d>: mov %eax,%esp
0x0004855e <0x0004855e>: mov %eax,%esp
0x0004855f <0x0004855f>: mov %eax,%esp
0x00048560 <0x00048560>: mov %eax,%esp
0x00048561 <0x00048561>: add %eax,%esp
0x00048562 <0x00048562>: add %eax,%esp
0x00048563 <0x00048563>: shr %eax,%esp
0x00048564 <0x00048564>: shr %eax,%esp
0x00048565 <0x00048565>: sub %eax,%esp
0x00048566 <0x00048566>: movl $0x804865e,%esp
0x00048567 <0x00048567>: call 0x00048534 <printf@plt>
0x00048568 <0x00048568>: movl $0x8048677,%esp
0x00048569 <0x00048569>: call 0x00048534 <printf@plt>
0x00048570 <0x00048570>: lea -0x78(%ebp),%eax
0x00048571 <0x00048571>: mov %eax,0x4(%esp)
0x00048572 <0x00048572>: movl $0x8048682,%esp
0x00048573 <0x00048573>: call 0x00048534 <scanf@plt>
0x00048574 <0x00048574>: lea -0x78(%ebp),%eax
0x00048575 <0x00048575>: mov %eax,0x4(%esp)
0x00048576 <0x00048576>: mov %eax,%esp
0x00048577 <0x00048577>: call 0x00048534 <check>
0x00048578 <0x00048578>: mov $0x0,%eax
0x00048579 <0x00048579>: leave
0x0004857a <0x0004857a>: ret
End of assembler dump.
(gdb) x /s 0x00048682
0x00048682: "%s"
(gdb) 
```

11. The comparison is not in this function but there is a function call made for the function **check**. We will be adding a debugger for the function test using **tbreak check** and then continue execution using **continue** and then disassemble the code

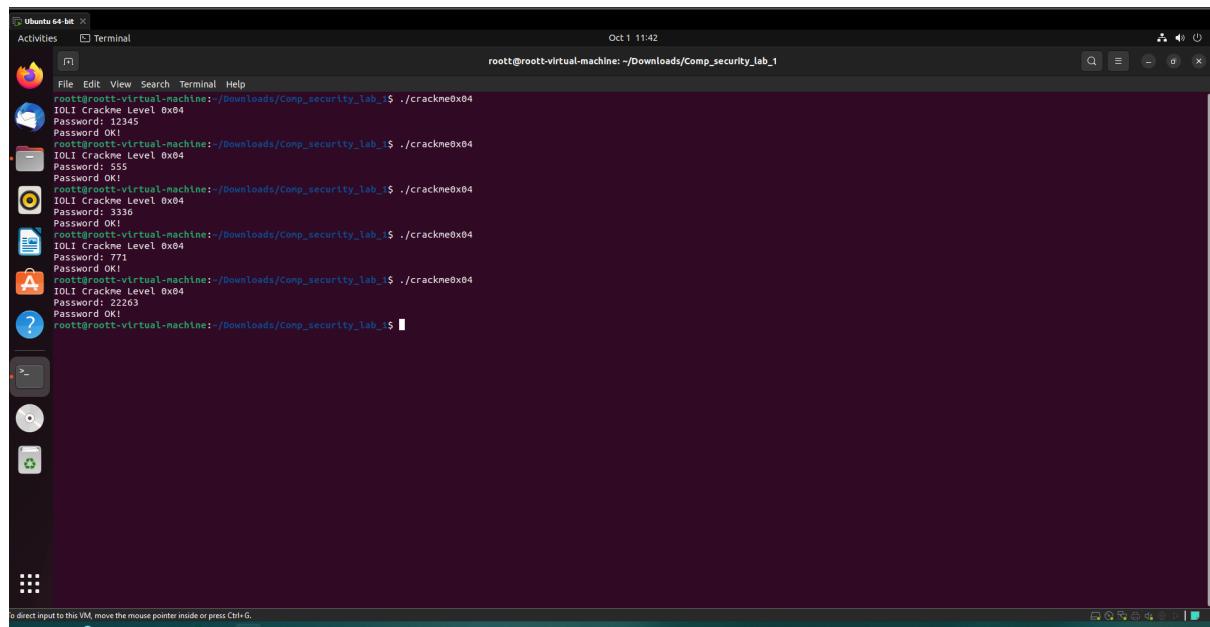
again using disas.



The screenshot shows the GDB debugger interface on an Ubuntu 64-bit system. The assembly dump for the `check()` function is displayed. The code includes temporary breakpoints at address 0x0804848a and 0x08048440. The assembly instructions show various memory operations, comparisons, and jumps, including a loop that seems to handle string input character by character. The debugger window has a terminal tab at the bottom where commands like `disas` and `step` were run.

12. There is a comparison made on line +26 but we can ignore this as it is comparing the current string length with 0.
13. As per the instructions, it seems like there is a loop being executed that is taking the string 1 character at a time, converting it to int using `sscanf` and then adding it to a variable and comparing it with 0xf hex or number 15 in decimal. This means the password is any string which has numbers that have a sum of 15.
14. Close the debugger using `quit` and launch the binary normally.
15. Enter the password.
16. Validate the result.

Output



The terminal window shows the execution of the `crackme0x04` binary. It prompts for a password and accepts "12345", "555", and "OKI" as valid inputs, indicating they sum up to 15. The window also shows other attempts like "771" and "223" which are rejected.

Binary 6 - Crackme0x05 - Password: Any number password with sum of digits = 16 and last digit is even. For ex 88 or 772 or 33334

Steps

1. Open the Binary file in debug mode using **gdb ./<path to file>**
 2. Run the binary using **run**
 3. Interrupt using **ctrl+c**
 4. Get the backtrace of code using **bt**
 5. Add a breakpoint on main using **tbreak <memory address of main>**
 6. Continue the code execution using **continue**
 7. Enter a password - **226655 and press enter**
 8. The screen should like like following

The screenshot shows a terminal window titled "Activities Terminal" on an Ubuntu 64-bit desktop environment. The terminal is running the command "gdb ./crackme0x05". The output shows the GDB prompt, the source code for crackme0x05, assembly dump, and assembly dump for function main. A temporary breakpoint is set at address 0x0804858a.

```
root@kali:~/Downloads/Comp_security_lab_1# gdb ./crackme0x05
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
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 "help" for information on how to use GDB.
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"...
Reading symbols from ./crackme0x05...
(No debugging symbols found in ./crackme0x05)
(gdb) run
Starting program: /home/root/Downloads/Comp_security_lab_1/crackme0x05
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[Crackme Level 0x05
Passing C
Program received signal SIGINT, Interrupt.
0xffff7c4549 in __kernel_vsyscall
(gdb) bt
#0 0xffff7c4549 in __kernel_vsyscall ()
#1 0xffff7d000f in read () from /lib/i386-linux-gnu/libc.so.6
#2 0xffff7df030 in __IO_file_underflow () from /lib/i386-linux-gnu/libc.so.6
#3 0xffff7df090 in __IO_defl_overflow () from /lib/i386-linux-gnu/libc.so.6
#4 0xffff7dd5e0 in ?? () from /lib/i386-linux-gnu/libc.so.6
#5 0xffff7d000e in __main () from /lib/i386-linux-gnu/libc.so.6
#6 0xffff7c4549 in __main ()
(gdb) bbreak *0x0804858a
Temporary breakpoint 1 at 0x0804858a
(gdb) c
Continuing.
226055

Temporary breakpoint 1, 0x0804858a in main ()
(gdb) disas
Dump of assembler code for function main:
0x0804858a <main>:    push    %rbp
0x0804858f <main+1>:   mov     %rsp,%rbp
0x08048591 <main+3>:   mov     %rbp,%esp
0x08048593 <main+5>:   sub    $0x10,%esp
```

- ## 9. Disassemble the binary using **disas**

10. Find the input format for **scanf**

The screenshot shows a terminal window titled "Ubuntu 64-bit" running on a virtual machine. The terminal displays assembly code from a debugger session, specifically the GDB debugger. The assembly code is for a function named "main". The code includes various instructions like push, mov, add, sub, and calls to functions like __printf_chk and __canskipplt. The debugger also shows temporary breakpoints and memory dump options.

```
(gdb) bt
#0 0x7fffcc5459 in _kernel_vsystcall ()
#1 0x7ffea0d7 in read () from /lib/i386-linux-gnu/libc.so.6
#2 0x77df0839 in __IO_file_underflow () from /lib/i386-linux-gnu/libc.so.6
#3 0x77df0997 in __IO_default_uflow () from /lib/i386-linux-gnu/libc.so.6
#4 0x77df0997 in __read_nocancel () from /lib/i386-linux-gnu/libc.so.6
#5 0x77dd8bb5 in scan () from /lib/i386-linux-gnu/libc.so.6
#6 0x0804858a in main ()
(gdb) bbreak *0x0804858a
Temporary breakpoint 1 at 0x0804858a
Continuing.
226655

Temporary breakpoint 1, 0x0804858a in main ()
(gdb) x/10s
Dump of assembler code for function main:
0x08048540 <+0>: push  %ebp
0x08048541 <+1>: mov   %esp,%ebp
0x08048543 <+3>: sub   $0x88,%esp
0x08048549 <+15>: and   $0xfffffff0,%esp
0x08048550 <+16>: mov   %eax,%eax
0x08048551 <+17>: add   $0xf,%eax
0x08048554 <+20>: add   $0xf,%eax
0x08048557 <+23>: shr  $0x4,%eax
0x08048558 <+26>: shr  $0x4,%eax
0x08048559 <+27>: sub   $0x8,%esp
0x0804855f <+31>: movl $0x0804868e,(%esp)
0x08048566 <+38>: call  __printf_chk(%esp)
0x0804856b <+43>: movl $0x80486a7,(%esp)
0x08048572 <+50>: call  __canskipplt(%esp)
0x08048577 <+55>: lea    %esp,%eax
0x08048580 <+58>: mov   %eax,%esp
0x0804857e <+62>: movl $0x80486b2,(%esp)
0x08048585 <+69>: call  __canskipplt(%esp)
=> 0x0804858a <+74>: lea    -0x78(%ebp),%eax
0x0804858d <+77>: mov   %eax,%esp
0x08048590 <+80>: call  __check(%esp)
0x08048593 <+85>: mov   $0x0,%eax
0x0804859a <+90>: leave 
0x0804859b <+91>: ret
End of assembler dump.
(gdb) x/5x 0x080486b2
0x080486b2 : "%
```

11. The comparison is not in this function but there is a function call made for the function **check**. We will be adding a debugger for the function test using **tbreak check** and then continue execution using **continue** and then disassemble the code again using **disas**.

The screenshot shows a GDB session running on an Ubuntu 64-bit system. The terminal window title is "root@roott-virtual-machine: ~/Downloads/Comp_security_lab_1". The assembly dump shows the code for the "check" function, which includes a temporary breakpoint at address 0x800484ce. The assembly code uses x86 instructions like push, mov, add, and cmp, with memory addresses and registers like %esp, %ebp, and %eax.

```
File Edit View Search Terminal Help
0x80048392 <+90>; leave
0x8004839b <+91>; ret
End of assembler dump.
(gdb) x /s 0x800486b2
0x800486b2:    db 73h
(gdb) bbreak check
Temporary breakpoint 2 at 0x800484ce
(gdb) c
Continuing.
Temporary breakpoint 2, 0x800484ce in check ()
(gdb) disas
Dump of assembler code for function check:
0x000484c8 <+0>; push %ebp
0x000484c9 <+1>; mov %ebp,%ebp
0x000484ca <+2>; sub $0x40,%esp
0x000484cc <+40>; movl $0xb0,-0x8(%ebp)
0x000484cd <+13>; movl $0xb0,-0xc(%ebp)
0x000484dc <+20>; movl $0xd(%ebp),%eax
0x000484df <+23>; mov %eax,(%esp)
0x000484e0 <+24>; call _scanf@plt <scanf@plt>
0x000484e7 <+31>; cmp %eax,-0xc(%ebp)
0x000484e8 <+34>; ja 0x00048532 <check+106>
0x000484ec <+36>; mov -0xc(%ebp),%eax
0x000484ef <+39>; add $0x8(%ebp),%eax
0x000484f2 <+42>; movbl $(%eax),%eax
0x000484f4 <+45>; movl $0x10000000,%eax
0x000484f8 <+48>; lea -0x4(%ebp),%eax
0x000484fb <+51>; mov %eax,0x0(%esp)
0x000484ff <+55>; movl $0x8048668,0x4(%esp)
0x00048507 <+63>; lea -0xd(%ebp),%eax
0x00048508 <+64>; mov %eax,(%esp)
0x0004850d <+69>; call _scanf@plt <scanf@plt>
0x00048512 <+74>; mov -0x4(%ebp),%edx
0x00048515 <+77>; lea -0x8(%ebp),%eax
0x00048518 <+80>; add %edx,(%eax)
0x0004851b <+83>; cmpl $0x10,-0xd(%ebp)
0x0004851e <+86>; jne 0x00048520 <check+99>
0x00048520 <+88>; mov 0x8(%ebp),%eax
0x00048523 <+91>; mov %eax,(%esp)
0x00048526 <+94>; call 0x80484840 <parell>
0x0004852b <+99>; lea -0xc(%ebp),%eax
0x00048530 <+102>; incl (%eax)
0x0004853b <+104>; jmp 0x80484864 <check+20>
0x00048532 <+106>; movl $0x8048679,(%esp)
....
```

12. There is a comparison made on line +26 but we can ignore this as it is comparing the current string length with 0.
 - 13. As per the instructions, it seems like there is a loop being executed that is taking the string 1 character at a time, converting it to int using sscanf and then adding it to a variable and comparing it with 0x10 hex or number 16 in decimal. This means the password is any string which has numbers that have a sum of 16.**
 14. After this, the control of the code moves to function parell
 15. Add a debugger on function parell and disassemble the code using das

```

0x080484807 <+63>: lea    -0xd(%ebp),%eax
0x080484850 <+66>: mov    %eax,(%esp)
0x08048485d <+69>: call   _scanf@plt
0x080484874 <+74>: mov    %eax,-0x4(%ebp)
0x08048515 <+77>: lea    -0x8(%ebp),%eax
0x08048518 <+80>: add    %edx,%eax
0x0804851a <+82>: cmpl  $0x10,-0x8(%ebp)
0x0804851e <+86>: jne    -0xb,%eax+99<
0x08048520 <+88>: mov    %eax,-0x4(%ebp)
0x08048523 <+91>: mov    %eax,%eax
0x08048526 <+94>: call   _printf@plt
0x0804852b <+99>: lea    -0xc(%ebp),%eax
0x0804852c <+102>: incl   (%eax)
0x0804852f <+105>: jmp   -0x4(%eax),<hect+20>
0x08048532 <+106>: movl  $0x80484879,(%esp)
0x08048539 <+113>: call   _printf@plt
0x0804853f <+118>: leave 
0x0804853f <+119>: ret
End of assembler dump.
(gdb) c
Continuing.
Breakpoint 2, 0x080484848a in parell ()
(gdb) disas
Dump of assembler code for function parell:
0x080484844 <+0>: push  %ebp
0x080484845 <+1>: mov   %esp,%ebp
0x080484847 <+3>: sub   $0x18,%esp
0x080484848 <+4>: lea    -0x4(%ebp),%eax
0x08048484d <+9>: mov   %eax,%eax
0x08048484f <+13>: movl  $0x80484808,%eax(%esp)
0x080484849 <+21>: movl  $0x80484808,%eax(%esp)
0x080484849 <+24>: mov   %eax,%eax
0x08048484a <+25>: call   _scanf@plt
0x08048484d <+32>: mov    -0x4(%ebp),%eax(%esp)
0x080484847 <+35>: and   $0x1,%eax
0x08048484a <+38>: test  %eax,%eax
0x08048484c <+40>: jne   -0x4(%ebp),<parell+6d>
0x08048484f <+42>: movl  $0x8048480b,(%esp)
0x080484851 <+44>: call   _printf@plt
0x080484854 <+45>: movl  $0x0,(%esp)
0x080484841 <+51>: call   _exit@plt
0x080484846 <+66>: leave 
0x080484846 <+67>: ret
End of assembler dump.

```

16.

17. On the line parell+35, there is an and operation with 1 which means that the last digit of the password must be even. So the final password should have a sum of digits as 16 and should be even so 772 works as a password but 277 does not.

18. Close the debugger using **quit** and launch the binary normally.

19. Enter the password.

20. Validate the result.

Output

```

root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 277
Password incorrect!
root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 772
Password OK!
root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 3334
Password OK!
root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 4444
Password OK!
root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ ./crackme0x05
IOLI Crackme Level 0x05
Password: 88
Password OK!
root@roott-virtual-machine:~/Downloads/Comp_security_lab_1$ 

```