System Programming Report

Document on GNU Debugger

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Roll No	Name of Student
2015UCP1338	UJESH MAURYA

Under the guidance of

Prof. Arka Prokash Mazumdar

Department of Computer Science and Engineering

Malaviya National Institute of Technology Jaipur

Jaipur, Rajasthan, India – 302 017

Even Semester 2017

1 WHAT IS A GNU DEBUGGER ??

A debugger is a program that runs other programs, allowing the user to exercise control over these programs, and to examine variables when problems arise.

GNU Debugger, which is also called gdb, is the most popular debugger for UNIX systems to debug C and C++ programs.

GNU Debugger helps you in getting information about the following:

1.1 If a core dump happened, then what statement or expression did the program crash on?

1.2 If an error occurs while executing a function, what line of the program contains the call to that function, and what are the parameters?

1.3 What are the values of program variables at a particular point during execution of the program?

GDB allows you to run the program up to a certain point, then stop and print out the values of certain variables at that point, or step through the program one line at a time and print out the values of each variable after executing each line. GDB uses a simple command line interface.

2 Various Commands in GDB

GDB offers a big list of commands, however the following commands are the ones used most frequently:

b main - Puts a breakpoint at the beginning of the program

b - Puts a breakpoint at the current line

b N - Puts a breakpoint at line N

b +N - Puts a breakpoint N lines down from the current line

b fn - Puts a breakpoint at the beginning of function ”fn”

d N - Deletes breakpoint number N

info break - list breakpoints

r - Runs the program until a breakpoint or error

c - Continues running the program until the next breakpoint or error

f - Runs until the current function is finished

s - Runs the next line of the program

s N - Runs the next N lines of the program n - Like s, but it does not step into functions

u N - Runs until you get N lines in front of the current line

p var - Prints the current value of the variable ”var”

bt - Prints a stack trace

u - Goes up a level in the stack

d - Goes down a level in the stack

q - Quits gdb

3 APPLICATION: Crack Password from executable file

3.1 Original Source code in C

```
#include <stdio.h>
#include <string.h>
int main(){
        char a[100];
        printf(”Enter the password to know the secret information”);
        scanf(”%s”, a);
        if (strcmp(a, ”mnit@12345”) == 0)
            printf(”Sherlock is one step ahead of Moriarty!!”);
        else
            printf(”Sorry !! Wrong Match.”);
        return 0;
}
```

3.2 STEP 1: Executable File

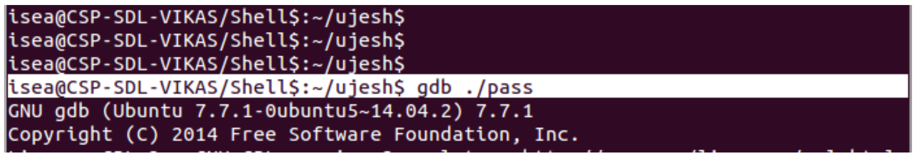
We will try to fetch the password defined in the source code without using this S source file; instead we will use it’s executable file.
To get the executable file of above program (pass.c), do this:

```
gcc pass.c -o pass
```

Now we have a executable file named ”pass”

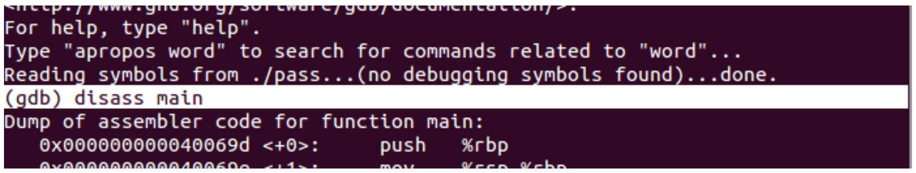
3.3 STEP 2: Using GDB

Now in order to get a object code from our executable file ”pass”, we will take help of GDB (GNU Debugger).
Run following command to get object code.



```
isea@CSP-SDL-VIKAS/Shell$:~/ujesh$
isea@CSP-SDL-VIKAS/Shell$:~/ujesh$
isea@CSP-SDL-VIKAS/Shell$:~/ujesh$
isea@CSP-SDL-VIKAS/Shell$:~/ujesh$ gdb ./pass
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
```

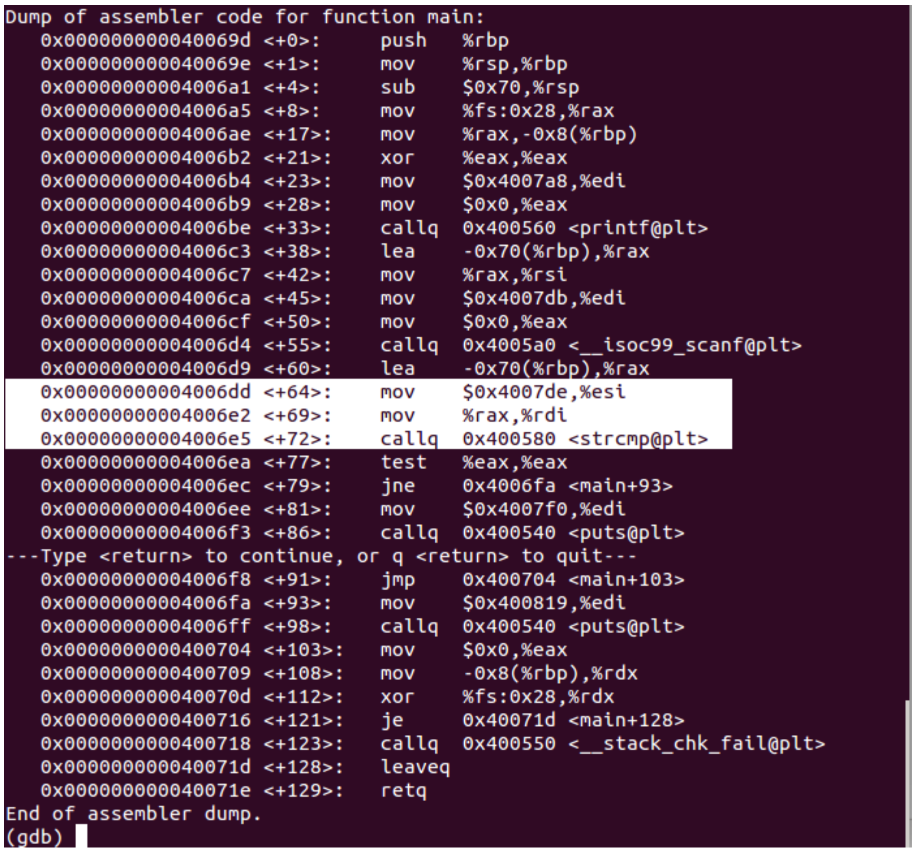
Following command will extract the object code:



```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./pass...(no debugging symbols found)...done.
(gdb) disass main
Dump of assembler code for function main:
   0x000000000040069d <+0>:     push   %rbp
```

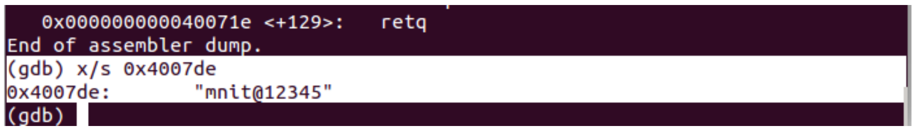
3.4 STEP 3: Searching Password

Since the program verifies the correctness of a input password, that implies that it must have the correct password inside it's code. That password must be a string and to match a string we need strcmp() function
So now our main target is to find the strcmp() function call in following assembly code.



```
Dump of assembler code for function main:
   0x000000000040069d <+0>:     push   %rbp
   0x000000000040069e <+1>:     mov    %rsp,%rbp
   0x00000000004006a1 <+4>:     sub    $0x70,%rsp
   0x00000000004006a5 <+8>:     mov    %fs:0x28,%rax
   0x00000000004006ae <+17>:    mov    %rax,-0x8(%rbp)
   0x00000000004006b2 <+21>:    xor    %eax,%eax
   0x00000000004006b4 <+23>:    mov    $0x4007a8,%edi
   0x00000000004006b9 <+28>:    mov    $0x0,%eax
   0x00000000004006be <+33>:    callq  0x400560 <printf@plt>
   0x00000000004006c3 <+38>:    lea    -0x70(%rbp),%rax
   0x00000000004006c7 <+42>:    mov    %rax,%rsi
   0x00000000004006ca <+45>:    mov    $0x4007db,%edi
   0x00000000004006cf <+50>:    mov    $0x0,%eax
   0x00000000004006d4 <+55>:    callq  0x4005a0 <__isoc99_scanf@plt>
   0x00000000004006d9 <+60>:    lea    -0x70(%rbp),%rax
   0x00000000004006dd <+64>:    mov    $0x4007de,%esi
   0x00000000004006e2 <+69>:    mov    %rax,%rdi
   0x00000000004006e5 <+72>:    callq  0x400580 <strcmp@plt>
   0x00000000004006ea <+77>:    test   %eax,%eax
   0x00000000004006ec <+79>:    jne    0x4006fa <main+93>
   0x00000000004006ee <+81>:    mov    $0x4007f0,%edi
   0x00000000004006f3 <+86>:    callq  0x400540 <puts@plt>
---Type <return> to continue, or q <return> to quit---
   0x00000000004006f8 <+91>:    jmp    0x400704 <main+103>
   0x00000000004006fa <+93>:    mov    $0x400819,%edi
   0x00000000004006ff <+98>:    callq  0x400540 <puts@plt>
   0x0000000000400704 <+103>:   mov    $0x0,%eax
   0x0000000000400709 <+108>:   mov    -0x8(%rbp),%rdx
   0x000000000040070d <+112>:   xor    %fs:0x28,%rdx
   0x0000000000400716 <+121>:   je     0x40071d <main+128>
   0x0000000000400718 <+123>:   callq  0x400550 <__stack_chk_fail@plt>
   0x000000000040071d <+128>:   leaveq
   0x000000000040071e <+129>:   retq
End of assembler dump.
(gdb)
```

We can see that on address 0x00000000004006e5 there is call to strcmp() function. Hence both the strings must have passed to this function as an argument. Since the original password was declared static so it must have been moved from a memory address to a constant register. Thus we can conclude that on address 0x00000000004006dd the mov command actually moves the value of original password to register ESI. So to view the password we can run following command:



```
   0x000000000040071e <+129>:   retq
End of assembler dump.
(gdb) x/s 0x4007de
0x4007de:       "mnit@12345"
(gdb)
```

As we can see in the screenshot that password was stored in the address 0x4007de. And we extracted the value of that address using x/s <address>command.

So Password is ”mnit@12345”