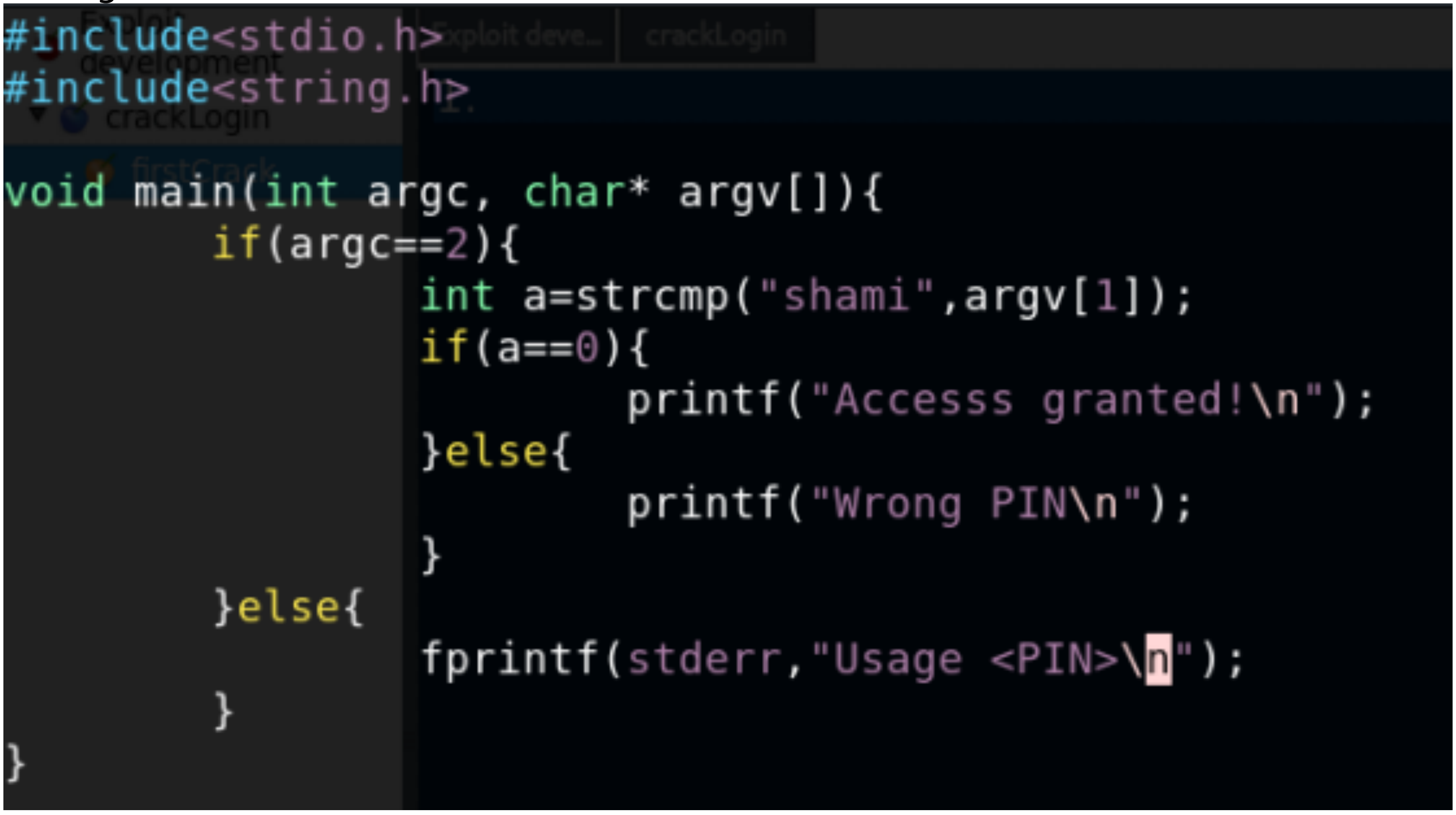
# *Exploit development*

## ***crackLogin***

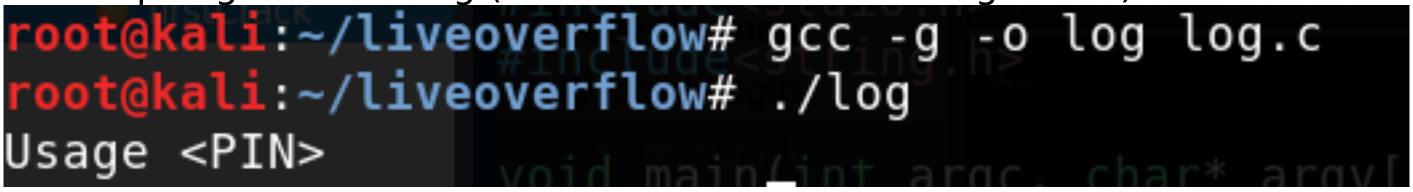
# firstCrack

1.Program to crack



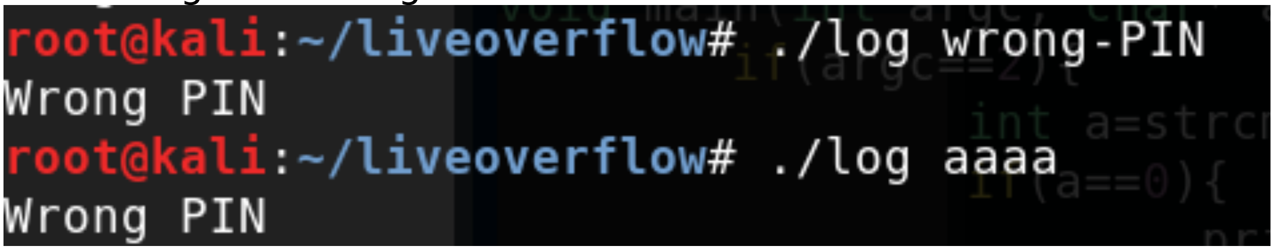
```
#include<stdio.h>
#include<string.h>

void main(int argc, char* argv[]){
        if(argc==2){
                int a=strcmp("shami",argv[1]);
                if(a==0){
                        printf("Accesss granted!\n");
                }else{
                        printf("Wrong PIN\n");
                }
        }else{
                fprintf(stderr,"Usage <PIN>\n");
        }
}
```

2. Compiling and running (Without command line argument)



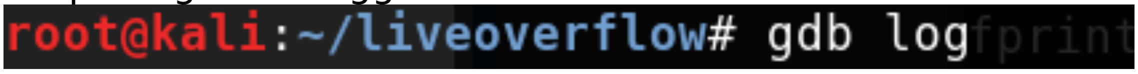
```
root@kali:~/liveoverflow# gcc -g -o log log.c
root@kali:~/liveoverflow# ./log
Usage <PIN>
```

3. Running with wrong PIN



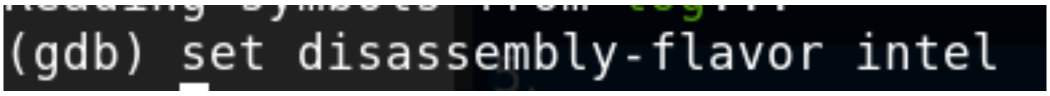
```
root@kali:~/liveoverflow# ./log wrong-PIN
Wrong PIN
root@kali:~/liveoverflow# ./log aaaa
Wrong PIN
```

4.Opening in debugger



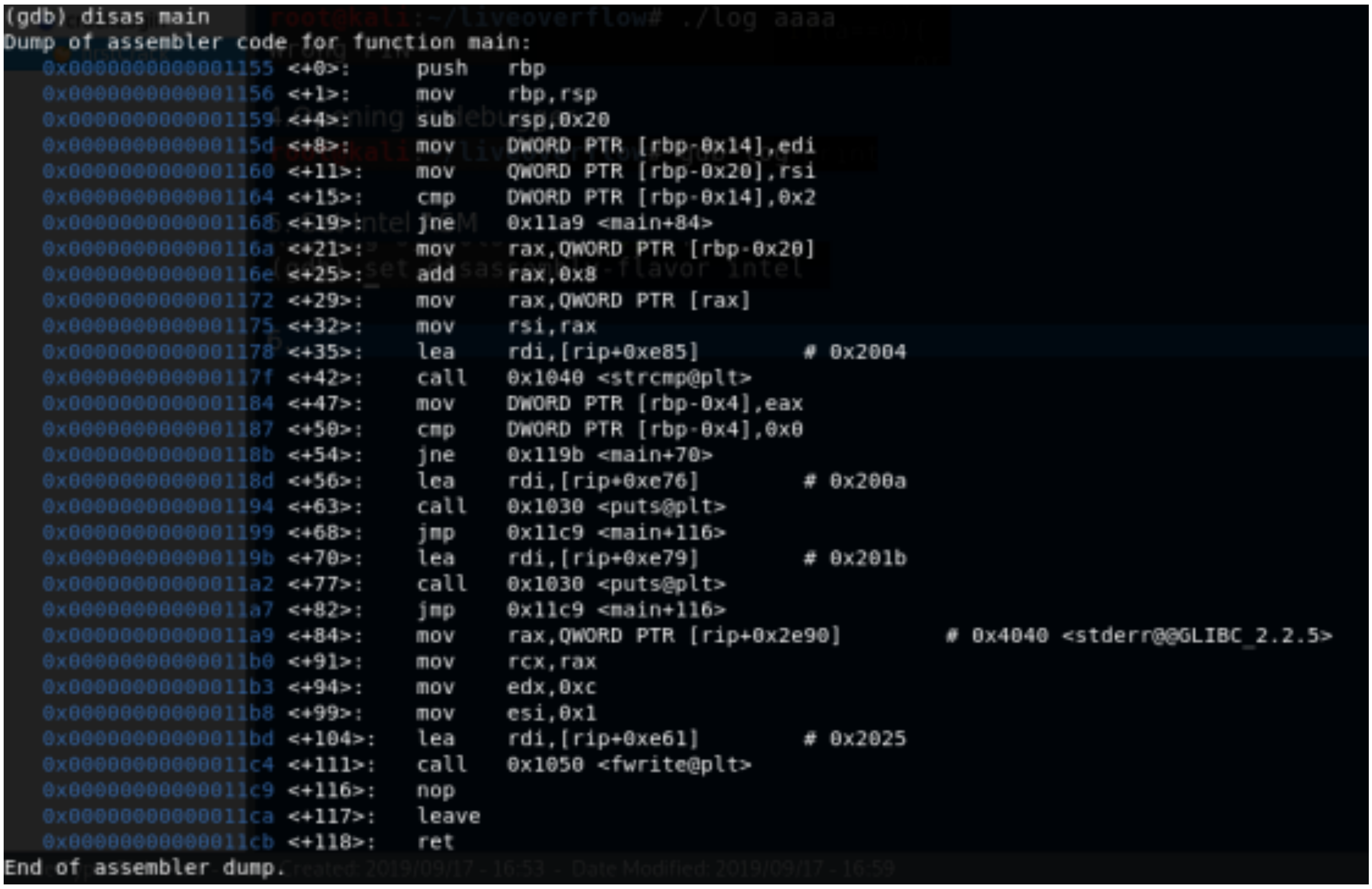
```
root@kali:~/liveoverflow# gdb log
```

5. Set Intel ASM



```
(gdb) set disassembly-flavor intel
```

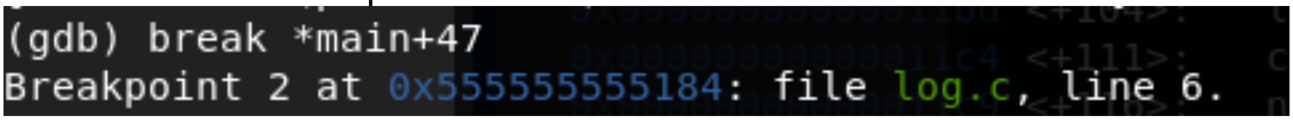
6.Disas main



```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000001155 <+0>:     push   rbp
   0x0000000000001156 <+1>:     mov    rbp,rsp
   0x0000000000001159 <+4>:     sub    rsp,0x20
   0x000000000000115d <+8>:     mov    DWORD PTR [rbp-0x14],edi
   0x0000000000001160 <+11>:    mov    QWORD PTR [rbp-0x20],rsi
   0x0000000000001164 <+15>:    cmp    DWORD PTR [rbp-0x14],0x2
   0x0000000000001168 <+19>:    jne    0x11a9 <main+84>
   0x000000000000116a <+21>:    mov    rax,QWORD PTR [rbp-0x20]
   0x000000000000116e <+25>:    add    rax,0x8
   0x0000000000001172 <+29>:    mov    rax,QWORD PTR [rax]
   0x0000000000001175 <+32>:    mov    rsi,rax
   0x0000000000001178 <+35>:    lea    rdi,[rip+0xe85]        # 0x2004
   0x000000000000117f <+42>:    call   0x1040 <strcmp@plt>
   0x0000000000001184 <+47>:    mov    DWORD PTR [rbp-0x4],eax
   0x0000000000001187 <+50>:    cmp    DWORD PTR [rbp-0x4],0x0
   0x000000000000118b <+54>:    jne    0x119b <main+70>
   0x000000000000118d <+56>:    lea    rdi,[rip+0xe76]        # 0x200a
   0x0000000000001194 <+63>:    call   0x1030 <puts@plt>
   0x0000000000001199 <+68>:    jmp    0x11c9 <main+116>
   0x000000000000119b <+70>:    lea    rdi,[rip+0xe79]        # 0x201b
   0x00000000000011a2 <+77>:    call   0x1030 <puts@plt>
   0x00000000000011a7 <+82>:    jmp    0x11c9 <main+116>
   0x00000000000011a9 <+84>:    mov    rax,QWORD PTR [rip+0x2e90]        # 0x4040 <stderr@@GLIBC_2.2.5>
   0x00000000000011b0 <+91>:    mov    rcx,rax
   0x00000000000011b3 <+94>:    mov    edx,0xc
   0x00000000000011b8 <+99>:    mov    esi,0x1
   0x00000000000011bd <+104>:   lea    rdi,[rip+0xe61]        # 0x2025
   0x00000000000011c4 <+111>:   call   0x1050 <fwrite@plt>
   0x00000000000011c9 <+116>:   nop
   0x00000000000011ca <+117>:   leave
   0x00000000000011cb <+118>:   ret
End of assembler dump.
```

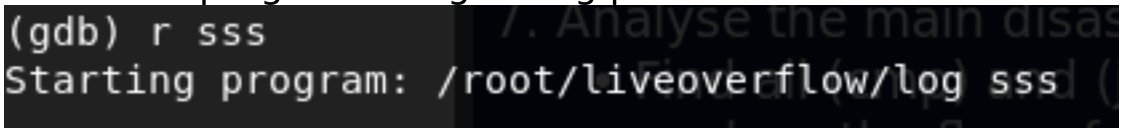
7. Analyse the main disassembely
   - Find all (cmp) and (jump) instructions
   - analyse the flow of program using the jump(jne in this case) functions
   - then find all the calls and draw an rough program flow
   - see that at main<+42> there is strcmp function> (This might be used for compairing the passwords!)

8. Set break points before the jumps or compare
   1) in this caes we will put it on main<+47>
   2)



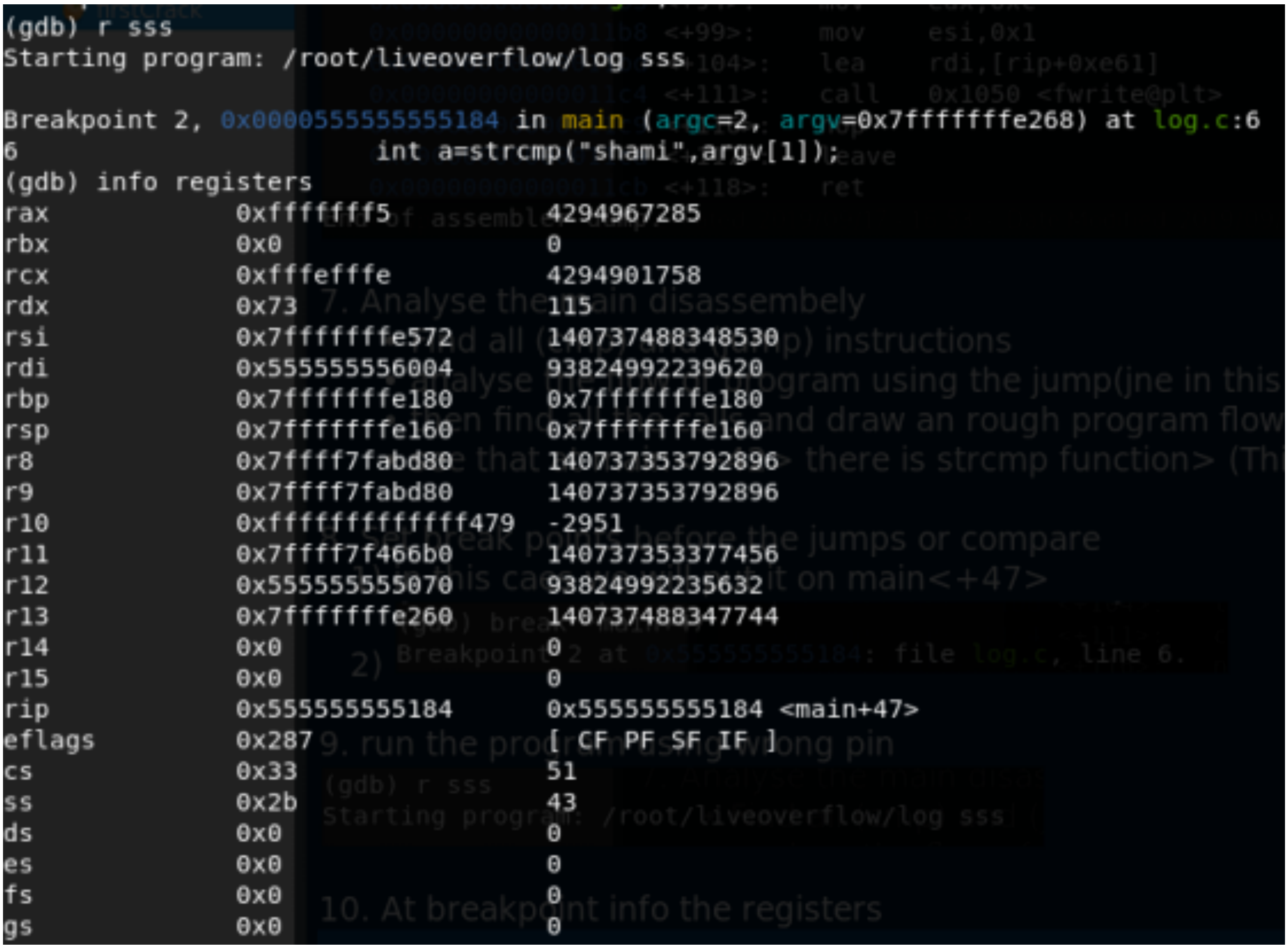
```
(gdb) break *main+47
Breakpoint 2 at 0x555555555184: file log.c, line 6.
```

9. run the program using wrong pin



```
(gdb) r sss
Starting program: /root/liveoverflow/log sss
```

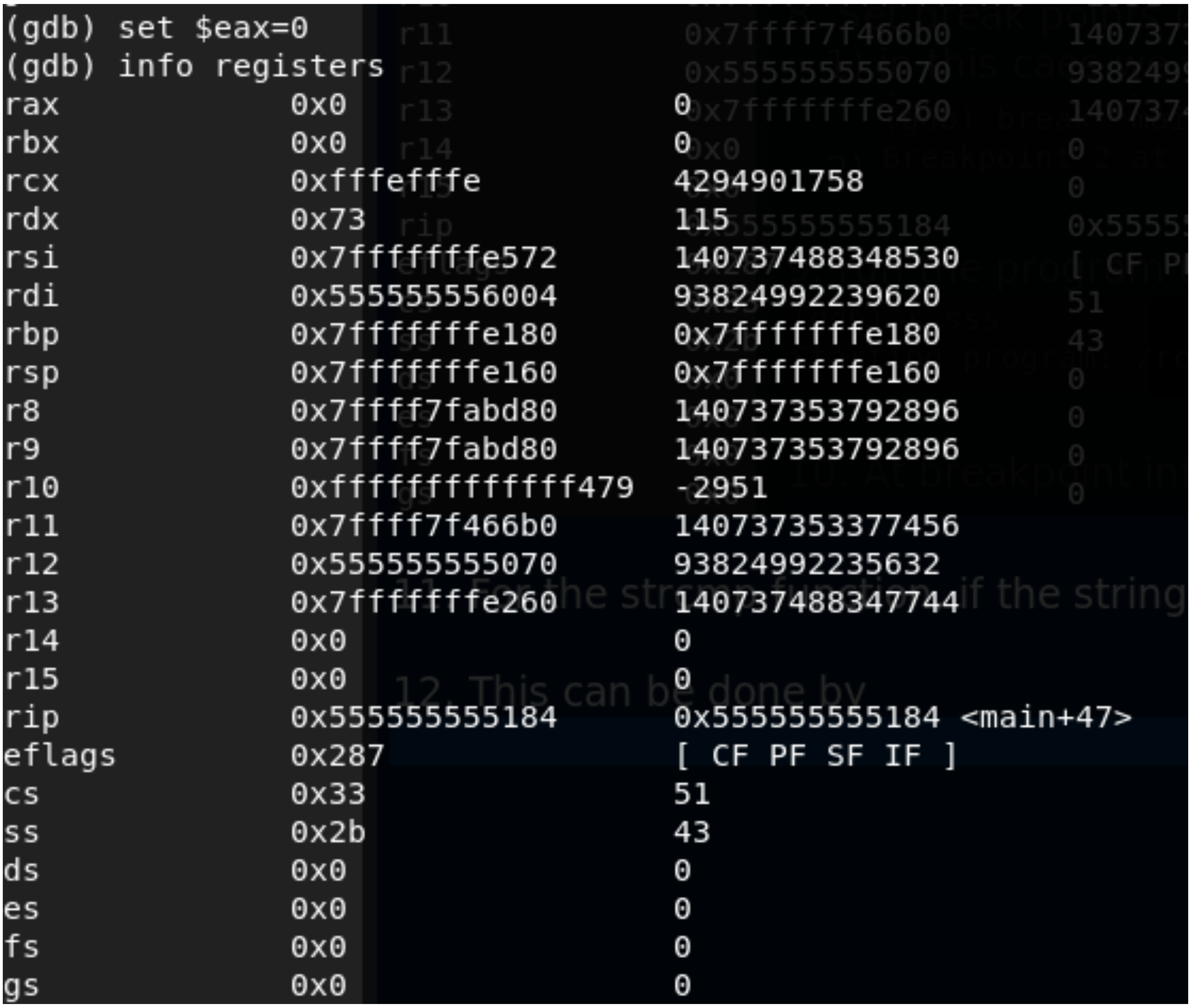
10. At breakpoint info the registers



```
(gdb) r sss
Starting program: /root/liveoverflow/log sss

Breakpoint 2, 0x0000555555555184 in main (argc=2, argv=0x7fffffffe268) at log.c:6
6                       int a=strcmp("shami",argv[1]);
(gdb) info registers
rax            0xfffffff5          4294967285
rbx            0x0                 0
rcx            0xfffefffe          4294901758
rdx            0x73                115
rsi            0x7fffffffe572      140737488348530
rdi            0x555555556004      93824992239620
rbp            0x7fffffffe180      0x7fffffffe180
rsp            0x7fffffffe160      0x7fffffffe160
r8             0x7ffff7fabd80      140737353792896
r9             0x7ffff7fabd80      140737353792896
r10            0xfffffffffffff479  -2951
r11            0x7ffff7f466b0      140737353377456
r12            0x555555555070      93824992235632
r13            0x7fffffffe260      140737488347744
r14            0x0                 0
r15            0x0                 0
rip            0x555555555184      0x555555555184 <main+47>
eflags         0x287               [ CF PF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

11. For the strcmp function, if the strings are equal output is 0 i.e, Zero flag is set (all 0's in eax register)

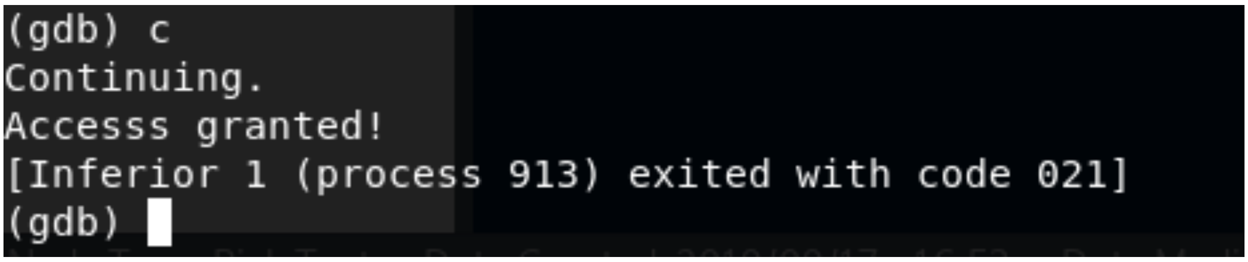
12. This can be done by



```
(gdb) set $eax=0
(gdb) info registers
rax            0x0                 0
rbx            0x0                 0
rcx            0xfffefffe          4294901758
rdx            0x73                115
rsi            0x7fffffffe572      140737488348530
rdi            0x555555556004      93824992239620
rbp            0x7fffffffe180      0x7fffffffe180
rsp            0x7fffffffe160      0x7fffffffe160
r8             0x7ffff7fabd80      140737353792896
r9             0x7ffff7fabd80      140737353792896
r10            0xfffffffffffff479  -2951
r11            0x7ffff7f466b0      140737353377456
r12            0x555555555070      93824992235632
r13            0x7fffffffe260      140737488347744
r14            0x0                 0
r15            0x0                 0
rip            0x555555555184      0x555555555184 <main+47>
eflags         0x287               [ CF PF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

- You can see that eax (rax) registers are set to 0
- Hence we explicitely set the zero flag
- Now continuing...

13. Continue



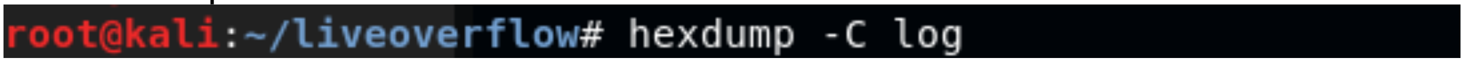
```
(gdb) c
Continuing.
Accesss granted!
[Inferior 1 (process 913) exited with code 021]
(gdb) 
```

14. Yay!! We see that access is granted!!

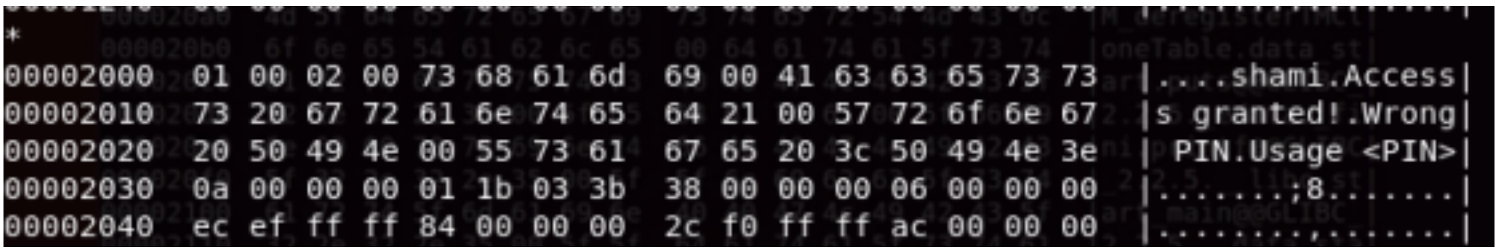
15. This is our first simple crack!

# *otherMethods*

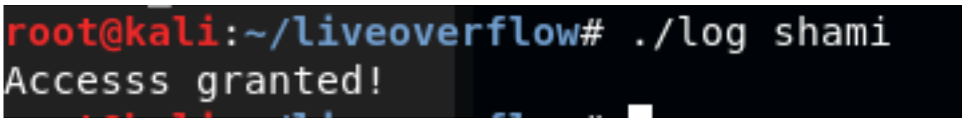
1. Hexdump



```
root@kali:~/liveoverflow# hexdump -C log
```



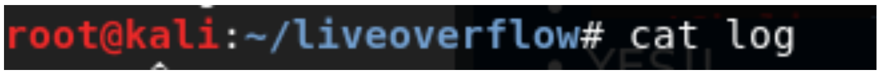
```
*
00002000  01 00 02 00 73 68 61 6d  69 00 41 63 63 65 73 73  |....shami.Access|
00002010  73 20 67 72 61 6e 74 65  64 21 00 57 72 6f 6e 67  |s granted!.Wrong|
00002020  20 50 49 4e 00 55 73 61  67 65 20 3c 50 49 4e 3e  | PIN.Usage <PIN>|
00002030  0a 00 00 00 01 1b 03 3b  38 00 00 00 06 00 00 00  |.......;8.......|
00002040  ec ef ff ff 84 00 00 00  2c f0 ff ff ac 00 00 00  |........,.......|
```

- Here we see that using hexdump, the strcmp string “shami” is printed
- By an educated guess we know that this could be key for program, lets try...
- 

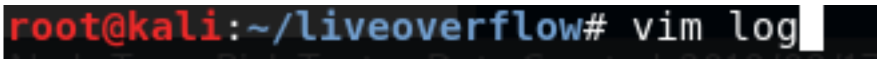
```
root@kali:~/liveoverflow# ./log shami
Accesss granted!
```

- YES!!

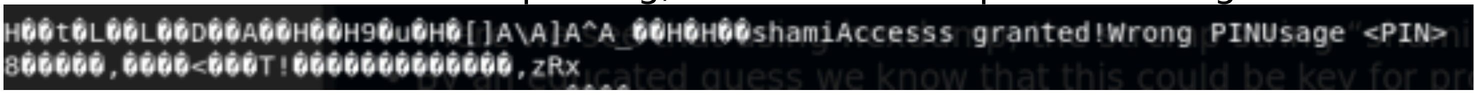
2. Opening executable using the text editors

- 

```
root@kali:~/liveoverflow# cat log
```

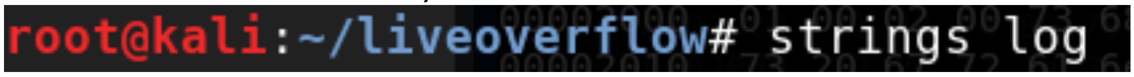
- 

```
root@kali:~/liveoverflow# vim log
```

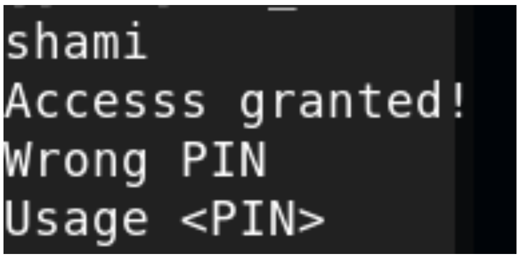
- In this both editors after inspecting, we can see the passwd string!!
- 

```
H��t�L��L��D��A��H��H9�u�H�[]A\A]A^A_��H�H��shamiAccesss granted!Wrong PINUsage <PIN>
8������,����<���T!��������������,zRx
```

3. Using strings utility (Strings→ prints all the priantable character sequences from executable on screen)

- 

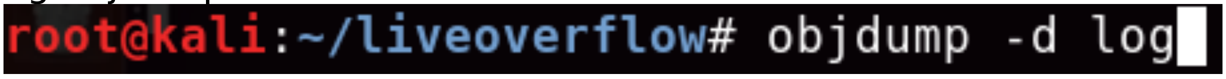
```
root@kali:~/liveoverflow# strings log
```

- 

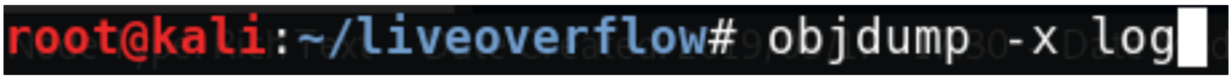
```
shami
Accesss granted!
Wrong PIN
Usage <PIN>
```

- Here is out password!

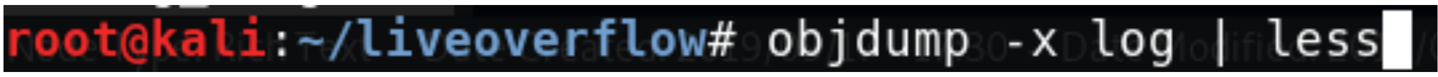
4. Using objdump!

-  -d for detai

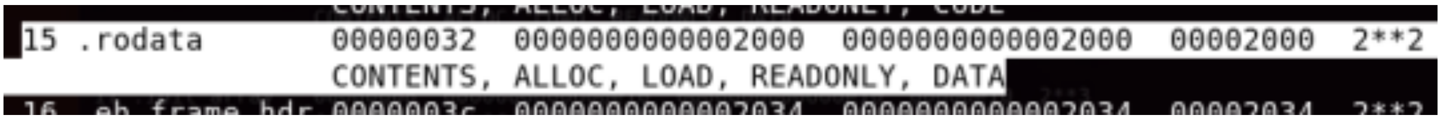
```
root@kali:~/liveoverflow# objdump -d log
```

-  -x for head

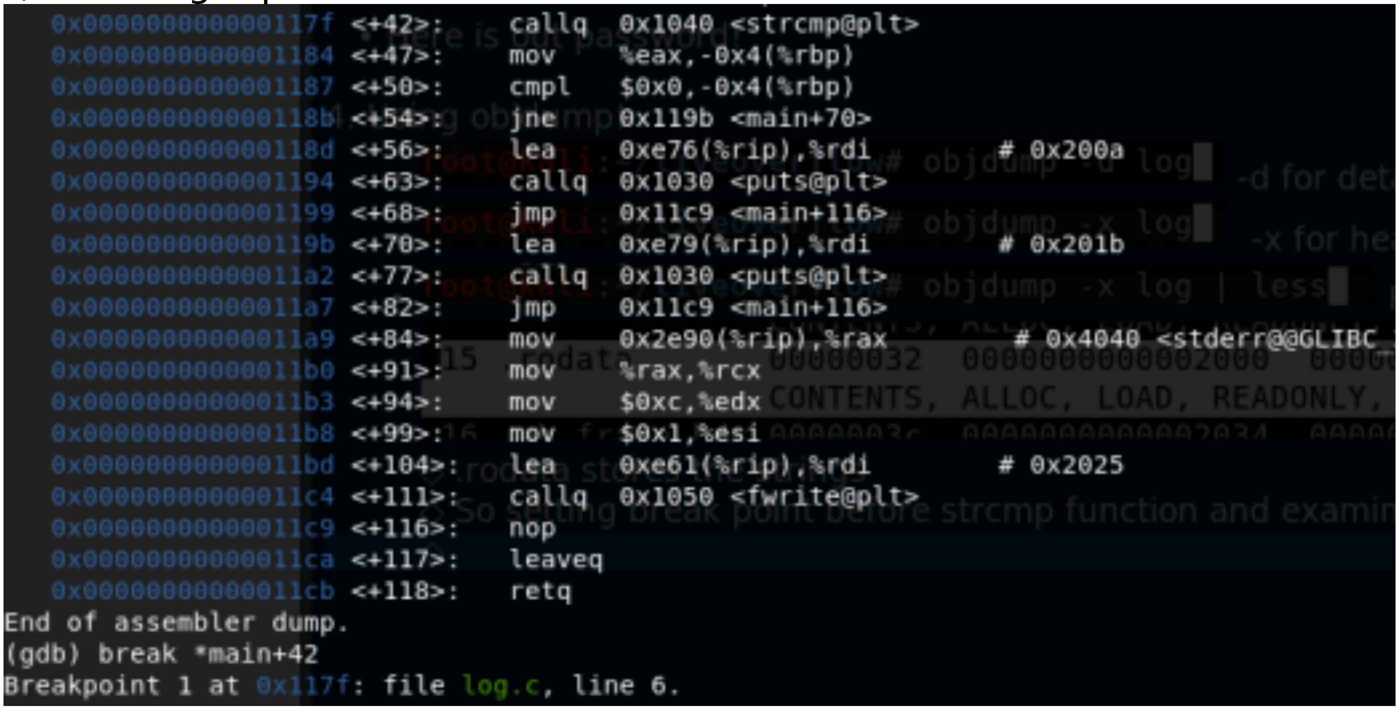
```
root@kali:~/liveoverflow# objdump -x log
```

- 

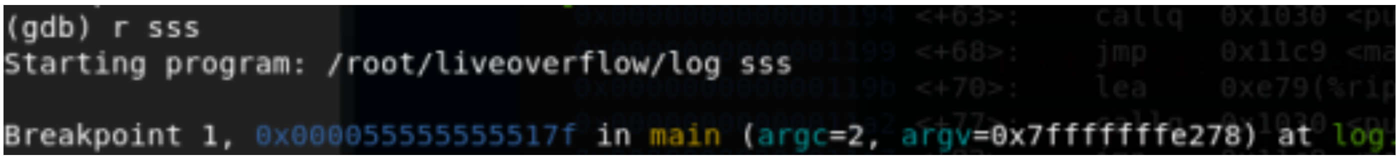
```
root@kali:~/liveoverflow# objdump -x log | less
```

- 

```
15 .rodata        00000032  0000000000002000  0000000000002000  00002000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
```

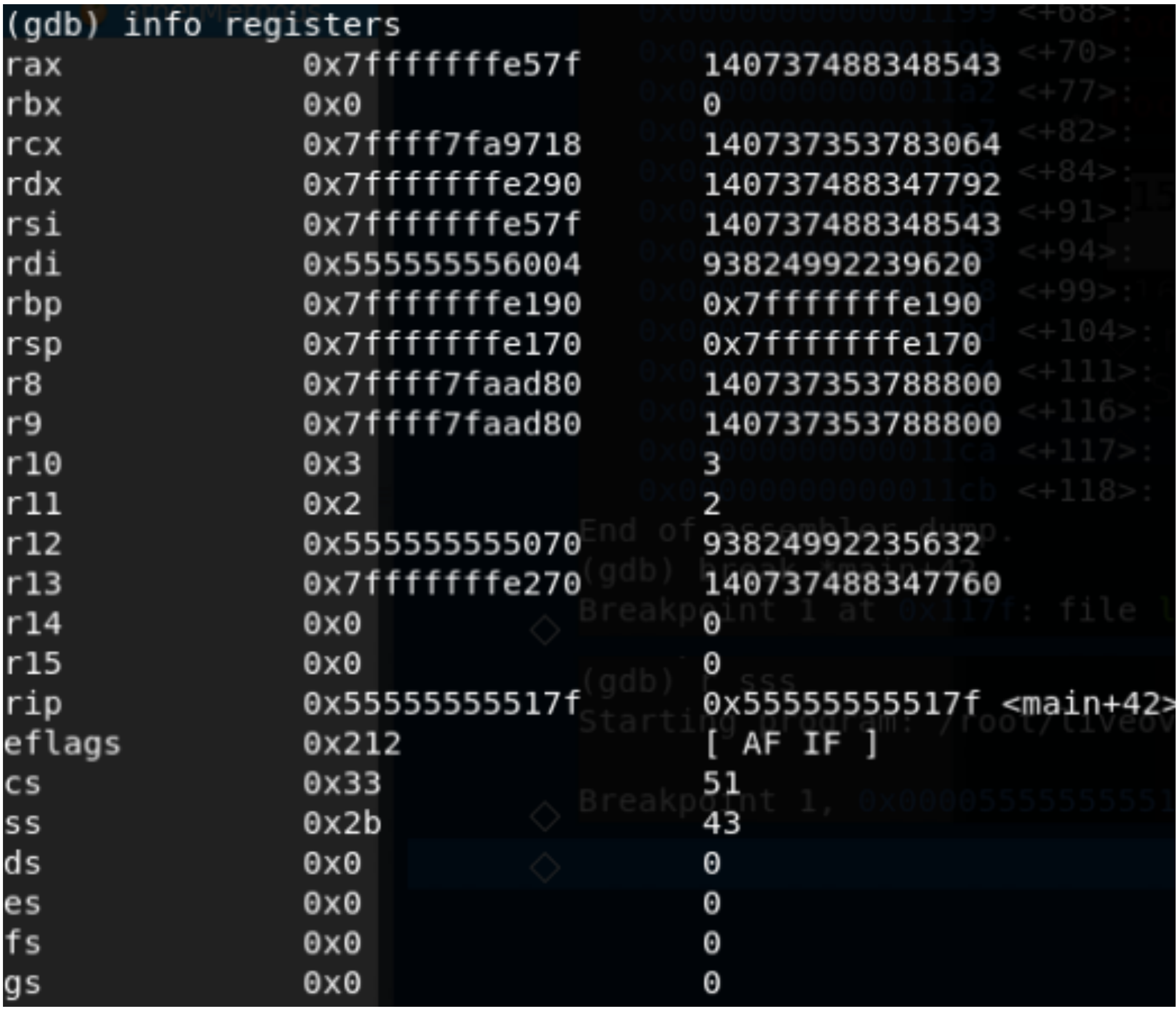
    ◇ .rodata stores the strings
    ◇ So setting break point before strcmp function and examining the registers, we can get passwd
    ◇ 

```
   0x000000000000117f <+42>:    callq  0x1040 <strcmp@plt>
   0x0000000000001184 <+47>:    mov    %eax,-0x4(%rbp)
   0x0000000000001187 <+50>:    cmpl   $0x0,-0x4(%rbp)
   0x000000000000118b <+54>:    jne    0x119b <main+70>
   0x000000000000118d <+56>:    lea    0xe76(%rip),%rdi        # 0x200a
   0x0000000000001194 <+63>:    callq  0x1030 <puts@plt>
   0x0000000000001199 <+68>:    jmp    0x11c9 <main+116>
   0x000000000000119b <+70>:    lea    0xe79(%rip),%rdi        # 0x201b
   0x00000000000011a2 <+77>:    callq  0x1030 <puts@plt>
   0x00000000000011a7 <+82>:    jmp    0x11c9 <main+116>
   0x00000000000011a9 <+84>:    mov    0x2e90(%rip),%rax        # 0x4040 <stderr@@GLIBC
   0x00000000000011b0 <+91>:    mov    %rax,%rcx
   0x00000000000011b3 <+94>:    mov    $0xc,%edx
   0x00000000000011b8 <+99>:    mov    $0x1,%esi
   0x00000000000011bd <+104>:   lea    0xe61(%rip),%rdi        # 0x2025
   0x00000000000011c4 <+111>:   callq  0x1050 <fwrite@plt>
   0x00000000000011c9 <+116>:   nop
   0x00000000000011ca <+117>:   leaveq
   0x00000000000011cb <+118>:   retq
End of assembler dump.
(gdb) break *main+42
Breakpoint 1 at 0x117f: file log.c, line 6.
```

    ◇ 

```
(gdb) r sss
Starting program: /root/liveoverflow/log sss

Breakpoint 1, 0x000055555555517f in main (argc=2, argv=0x7fffffffe278) at log
```

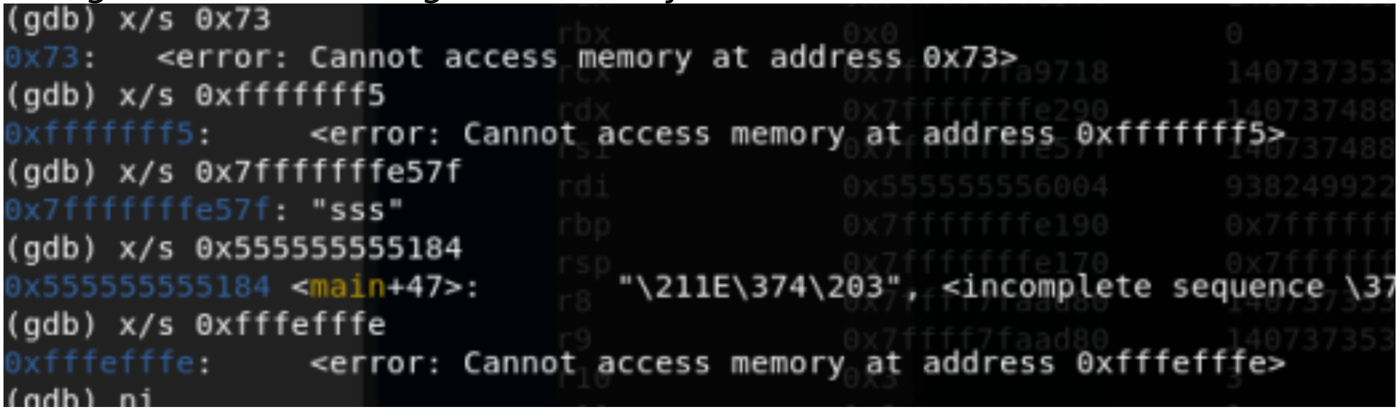
    ◇ Info registers→



```
(gdb) info registers
rax            0x7fffffffe57f      140737488348543
rbx            0x0                 0
rcx            0x7ffff7fa9718      140737353783064
rdx            0x7fffffffe290      140737488347792
rsi            0x7fffffffe57f      140737488348543
rdi            0x555555556004      93824992239620
rbp            0x7fffffffe190      0x7fffffffe190
rsp            0x7fffffffe170      0x7fffffffe170
r8             0x7ffff7faad80      140737353788800
r9             0x7ffff7faad80      140737353788800
r10            0x3                 3
r11            0x2                 2
r12            0x555555555070      93824992235632
r13            0x7fffffffe270      140737488347760
r14            0x0                 0
r15            0x0                 0
rip            0x55555555517f      0x55555555517f <main+42>
eflags         0x212               [ AF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
```

◇

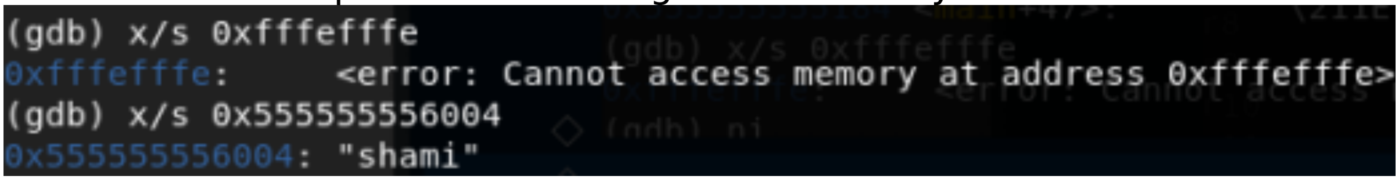
◇ Using x/s for examining the memory locations→



```
(gdb) x/s 0x73
0x73:   <error: Cannot access memory at address 0x73>
(gdb) x/s 0xfffffff5
0xfffffff5:     <error: Cannot access memory at address 0xfffffff5>
(gdb) x/s 0x7fffffffe57f
0x7fffffffe57f: "sss"
(gdb) x/s 0x555555555184
0x555555555184 <main+47>:       "\211E\374\203", <incomplete sequence \37
(gdb) x/s 0xfffefffe
0xfffefffe:     <error: Cannot access memory at address 0xfffefffe>
(gdb) ni
```

◇

◇ atlast findind the passwd on edi register's memory addr→

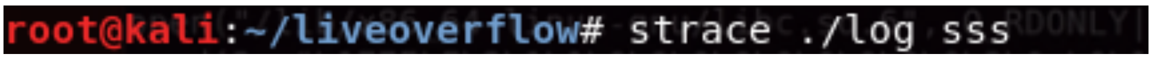


```
(gdb) x/s 0xfffefffe
0xfffefffe:     <error: Cannot access memory at address 0xfffefffe>
(gdb) x/s 0x555555556004
0x555555556004: "shami"
```

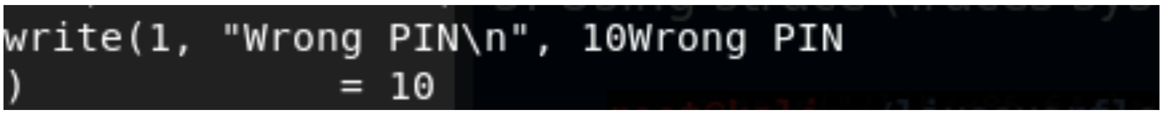
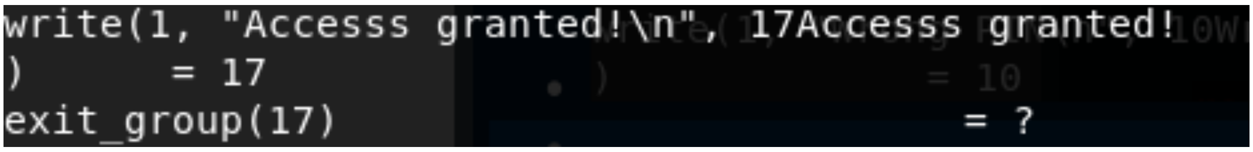
◇

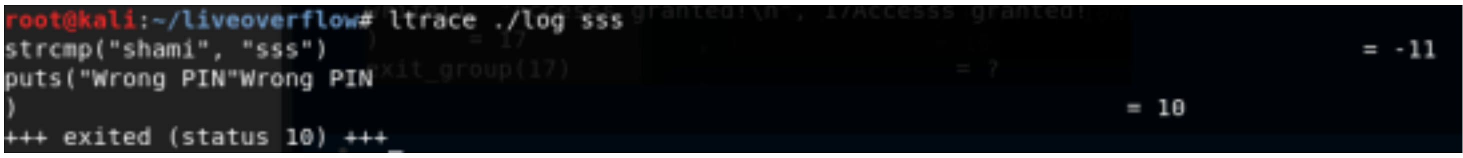
◇ Yay we found the string!!

5. Using strace (Traces syscall in program) and lstrace (traces lib functions)

- 

```
root@kali:~/liveoverflow# strace ./log sss
```

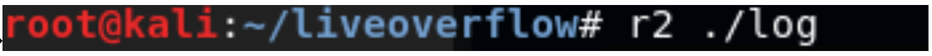
- 
```
write(1, "Wrong PIN\n", 10Wrong PIN
)                  = 10
```
- 
```
write(1, "Accesss granted!\n", 17Accesss granted!
)        = 17
exit_group(17)                              = ?
```

- 
```
root@kali:~/liveoverflow# ltrace ./log sss
strcmp("shami", "sss")                                                        = -11
puts("Wrong PIN"Wrong PIN
)                                                                     = 10
+++ exited (status 10) +++
```
- Here you can see the strcmp function used for comparing and we have the passwd!

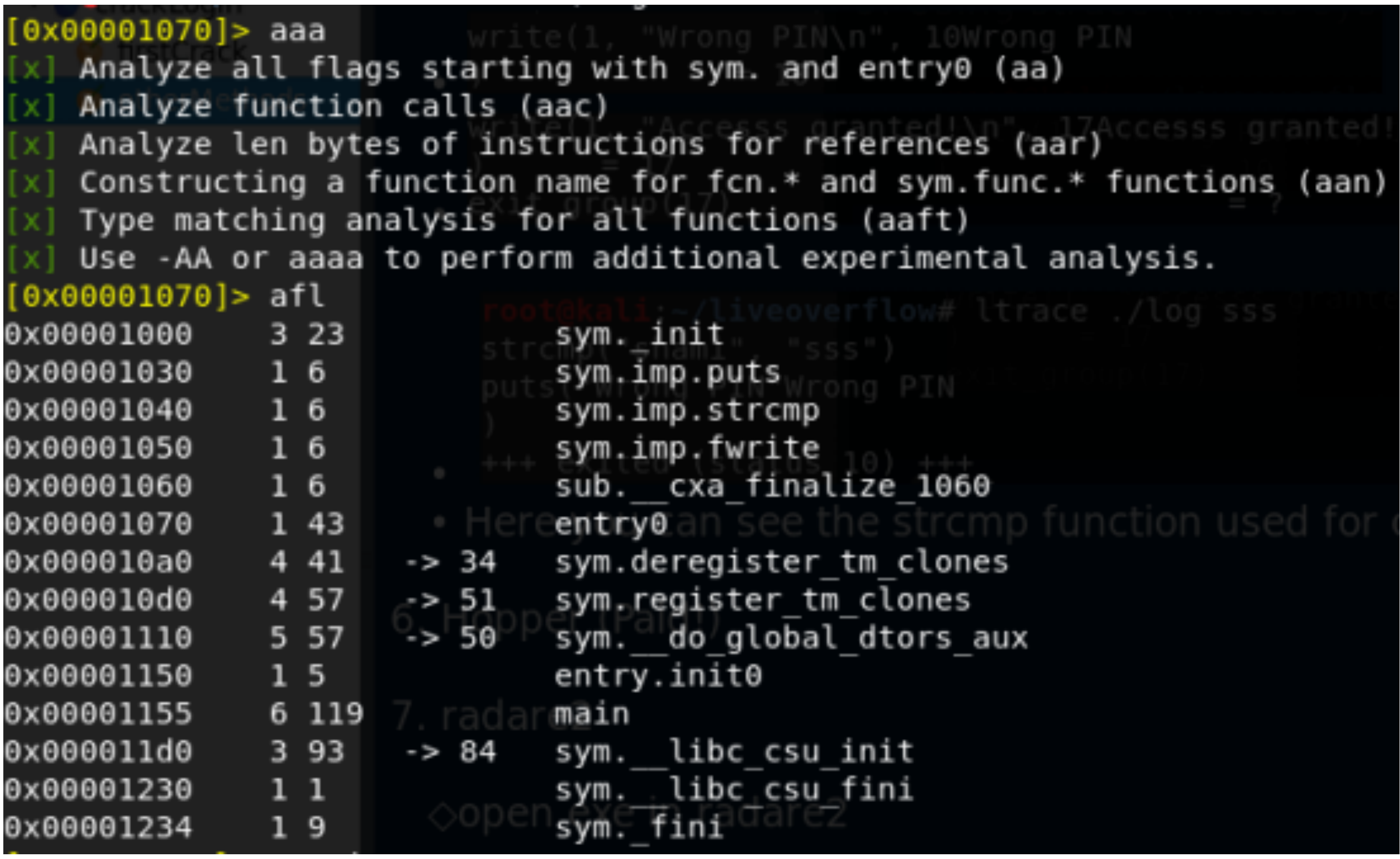
6. Hopper (Paid!)

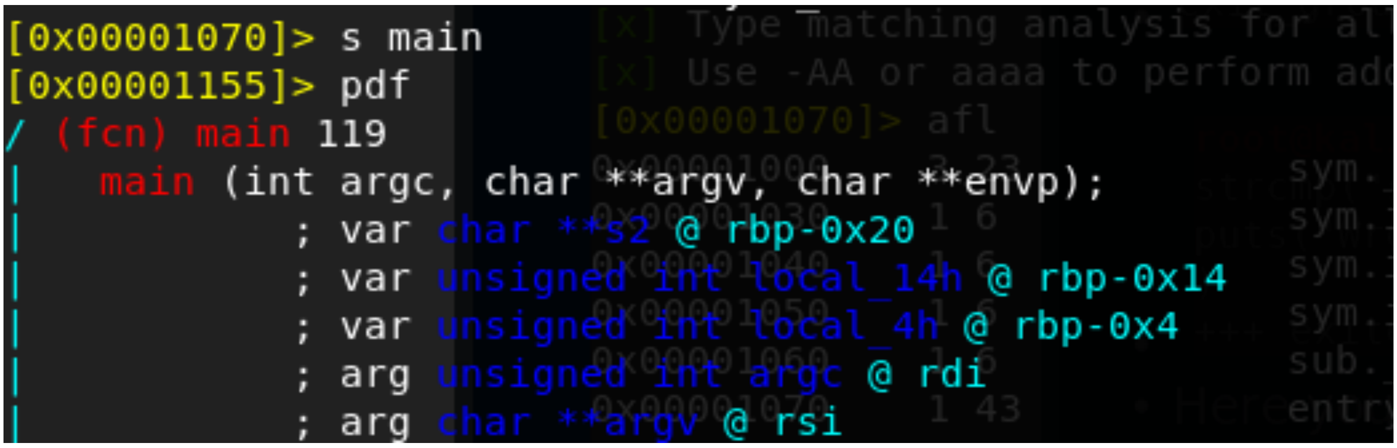
7. radare2

◇open exe in radare2

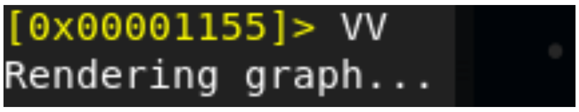
◇
```
root@kali:~/liveoverflow# r2 ./log
```

- use two commands "aaa" and "afl"

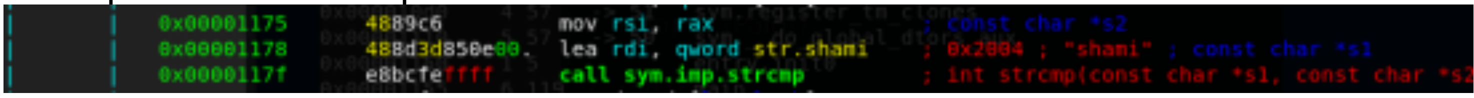
- 
```
[0x00001070]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Type matching analysis for all functions (aaft)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x00001070]> afl
0x00001000    3 23           sym._init
0x00001030    1 6            sym.imp.puts
0x00001040    1 6            sym.imp.strcmp
0x00001050    1 6            sym.imp.fwrite
0x00001060    1 6            sub.__cxa_finalize_1060
0x00001070    1 43           entry0
0x000010a0    4 41   -> 34   sym.deregister_tm_clones
0x000010d0    4 57   -> 51   sym.register_tm_clones
0x00001110    5 57   -> 50   sym.__do_global_dtors_aux
0x00001150    1 5            entry.init0
0x00001155    6 119          main
0x000011d0    3 93   -> 84   sym.__libc_csu_init
0x00001230    1 1            sym.__libc_csu_fini
0x00001234    1 9            sym._fini
```
- go to main function and display the diassembely

- 

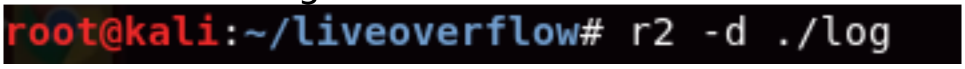
```
[0x00001070]> s main
[0x00001155]> pdf
/ (fcn) main 119
|   main (int argc, char **argv, char **envp);
|           ; var char **s2 @ rbp-0x20
|           ; var unsigned int local_14h @ rbp-0x14
|           ; var unsigned int local_4h @ rbp-0x4
|           ; arg unsigned int argc @ rdi
|           ; arg char **argv @ rsi
```

- VV for GUI
- 

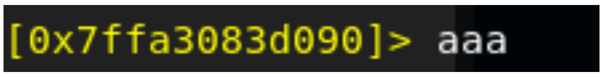
```
[0x00001155]> VV
Rendering graph...
```

- usinf pdf we can see password in clear text
- 

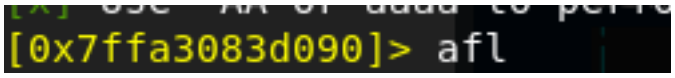
```
|       |   0x00001175      4889c6         mov rsi, rax                 ; const char *s2
|       |   0x00001178      488d3d850e00.  lea rdi, qword str.shami     ; 0x2004 ; "shami" ; const char *s1
|       |   0x0000117f      e8bcfeffff     call sym.imp.strcmp          ; int strcmp(const char *s1, const char *s2
```

- Start r2 in debug mode with “-d”
- 

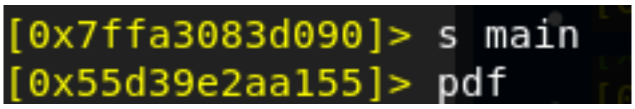
```
root@kali:~/liveoverflow# r2 -d ./log
```

- Enter all commands→
- 

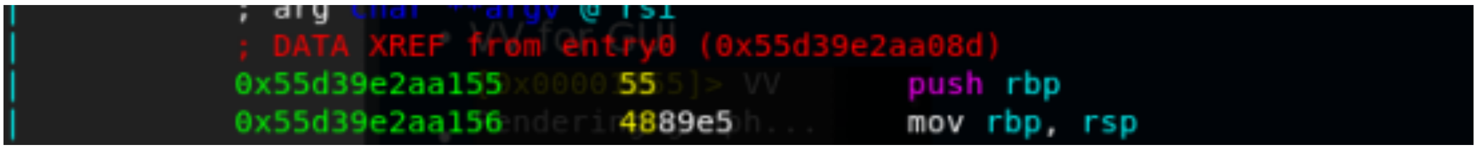
```
[0x7ffa3083d090]> aaa
```

- 

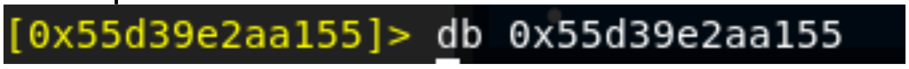
```
[0x7ffa3083d090]> afl
```

- 

```
[0x7ffa3083d090]> s main
[0x55d39e2aa155]> pdf
```

- 

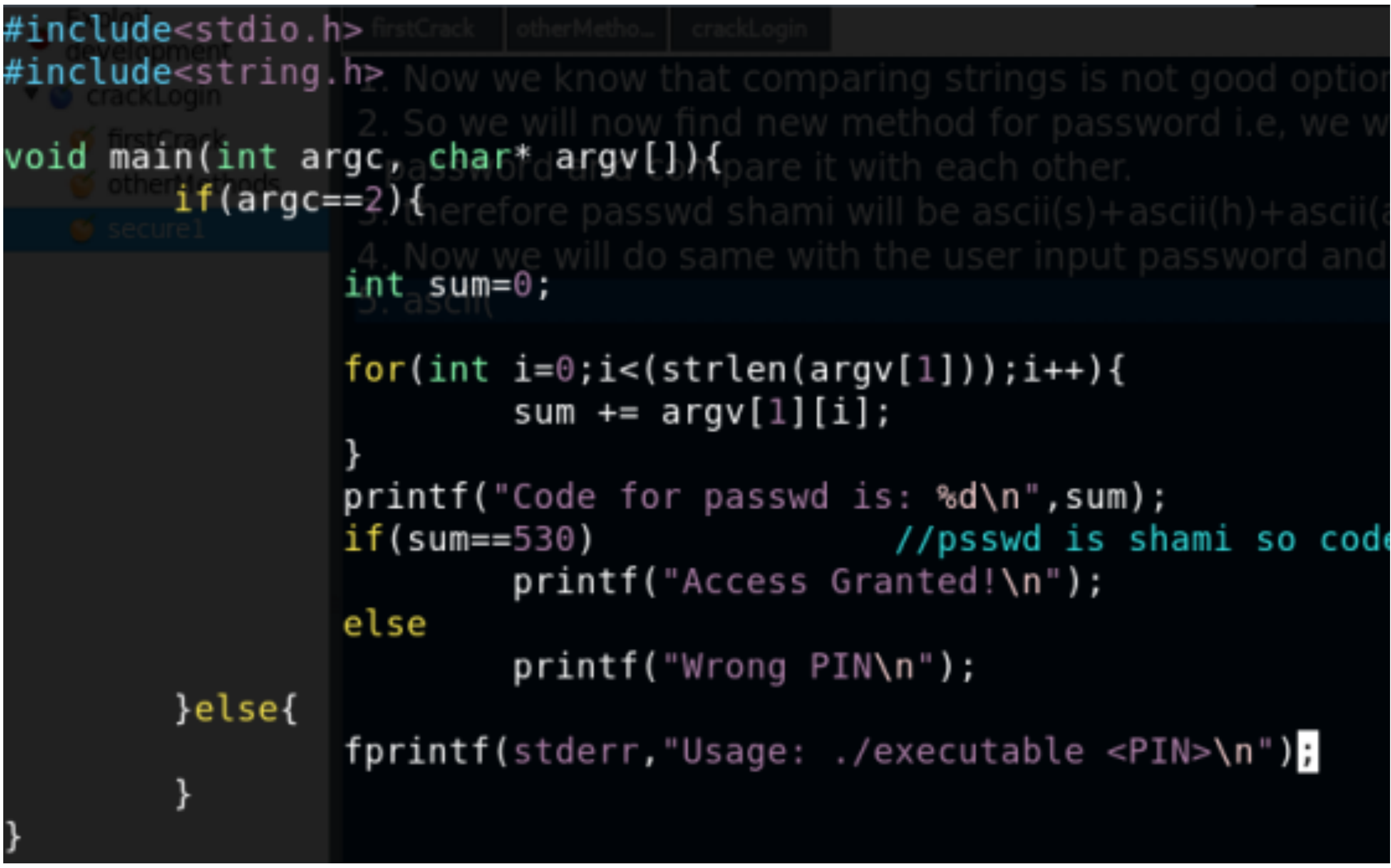
```
|           ; DATA XREF from entry0 (0x55d39e2aa08d)
|           0x55d39e2aa155      55             push rbp
|           0x55d39e2aa156      4889e5         mov rbp, rsp
```

- Break point at start
- 

```
[0x55d39e2aa155]> db 0x55d39e2aa155
```

- Enter into visual mode by VV
- Go to command mode by “:”
- Enter "dc" for continuing or running the program
- using “s” we can run one instruction at a time and “Shift s” does not stepps into functions

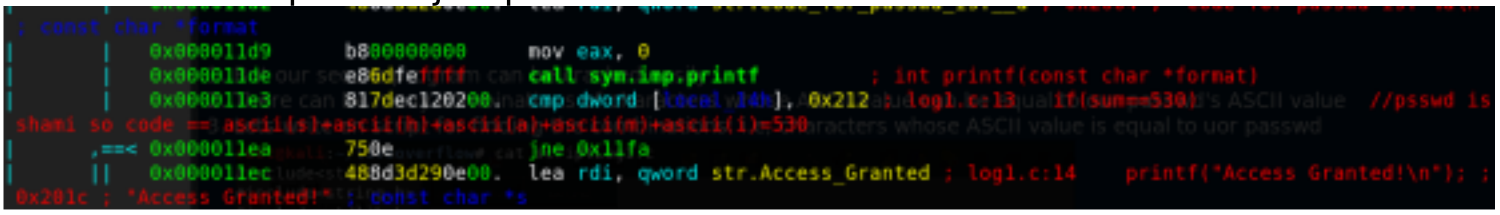
## *secure1*

1. Now we know that comparing strings is not good option
2. So we will now find new method for password i.e, we will add ascii values of the characters of password and user entered password and compare it with each other.
3. therefore passwd shami will be ascii(s)+ascii(h)+ascii(a)+ascii(m)+ascii(i) = 530
4. Now we will do same with the user input password and compare the code with 530
5. 

```
#include<stdio.h>
#include<string.h>

void main(int argc, char* argv[]){
        if(argc==2){

                int sum=0;

                for(int i=0;i<(strlen(argv[1]));i++){
                        sum += argv[1][i];
                }
                printf("Code for passwd is: %d\n",sum);
                if(sum==530)                    //psswd is shami so code
                        printf("Access Granted!\n");
                else
                        printf("Wrong PIN\n");
        }else{
                fprintf(stderr,"Usage: ./executable <PIN>\n");
        }
}
```

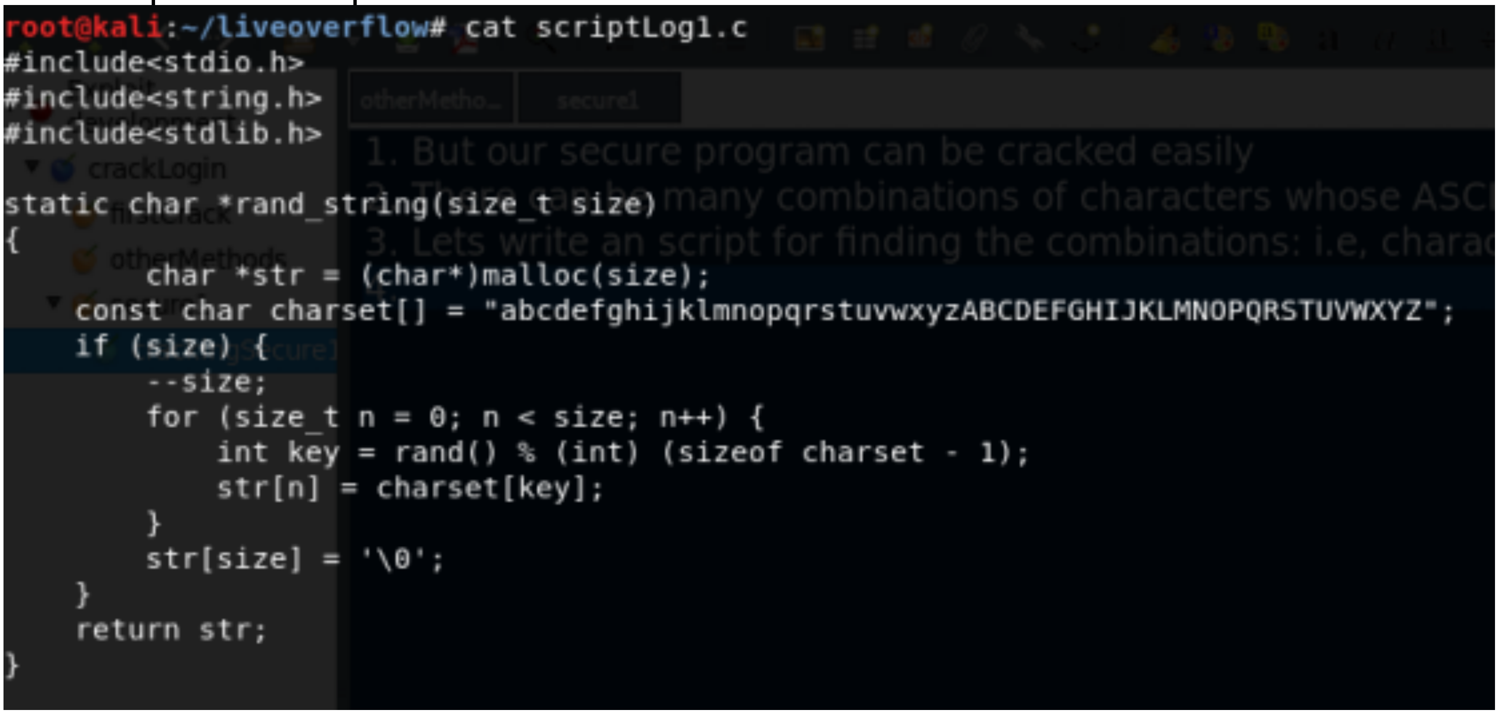
# crackingSecure1

• Now using debugger or radare2 we can analyse the binary of our program and find the compare or jump statements

• 

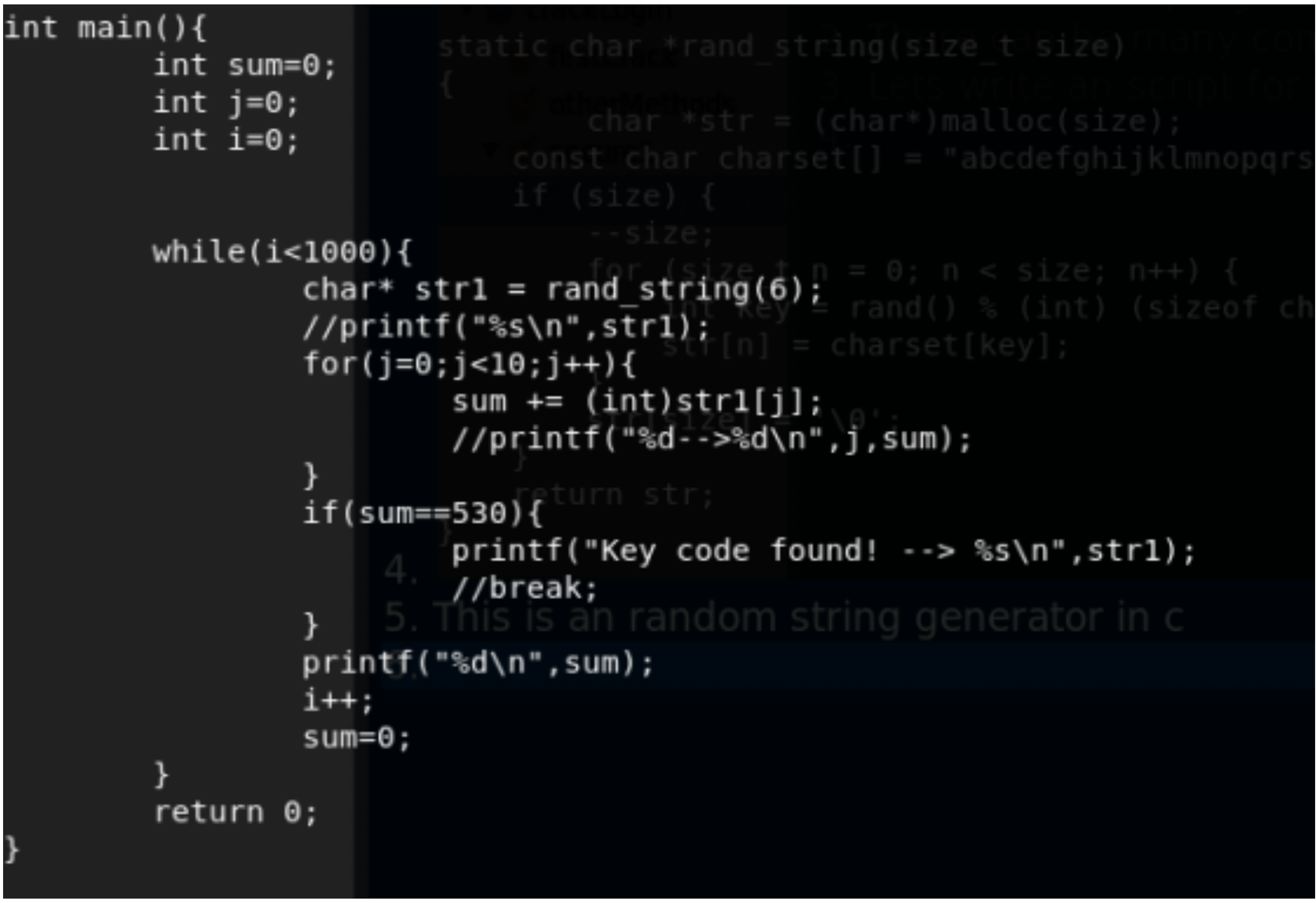
```
; const char *format
|        |   0x000011d9      b800000000     mov eax, 0
|        |   0x000011de      e86dfeffff     call sym.imp.printf        ; int printf(const char *format)
|        |   0x000011e3      817dec12020000. cmp dword [local_14h], 0x212 ; logl.c:13 if(sum==530)     //pswd is
shami so code == ascii(s)+ascii(h)+ascii(a)+ascii(m)+ascii(i)=530
|      ,==< 0x000011ea      750e           jne 0x11fa
|      ||   0x000011ec      488d3d290e00.  lea rdi, qword str.Access_Granted ; logl.c:14    printf("Access Granted!\n"); ;
0x201c ; "Access Granted!" ; const char *s
```

• Here above the Access_Granted string we can find the HEX number 0x212 which is decimal 530, which is our decoded value for the passwd

1. But our secure program can be cracked easily
2. There can be many combinations of characters whose ASCII value can be equal to our passwd's ASCII value
3. Lets write an script for finding the combinations: i.e, characters whose ASCII value is equal to uor passwd
4. 

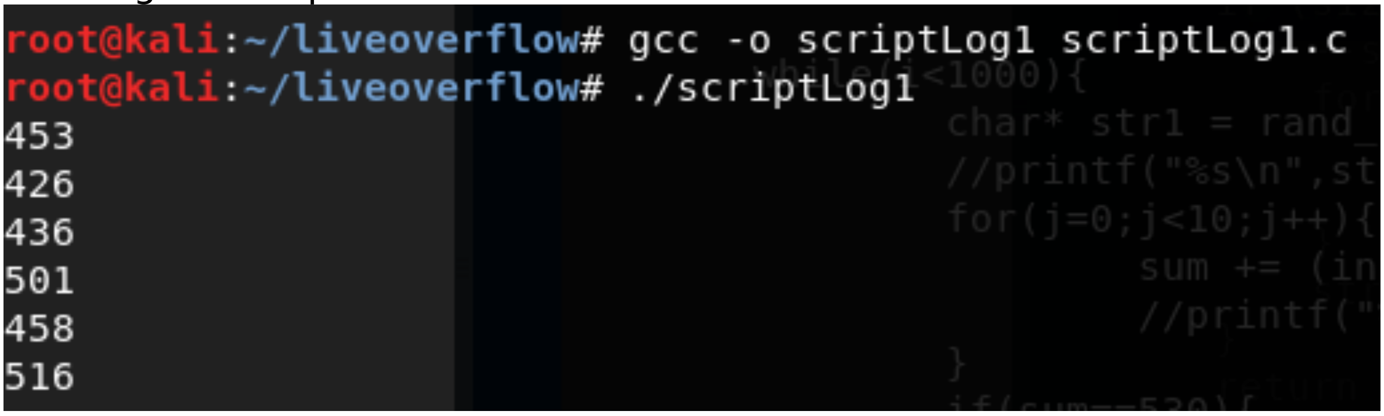
```
root@kali:~/liveoverflow# cat scriptLog1.c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

static char *rand_string(size_t size)
{
        char *str = (char*)malloc(size);
    const char charset[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
    if (size) {
        --size;
        for (size_t n = 0; n < size; n++) {
            int key = rand() % (int) (sizeof charset - 1);
            str[n] = charset[key];
        }
        str[size] = '\0';
    }
    return str;
}
```

5. This is an random string generator in c



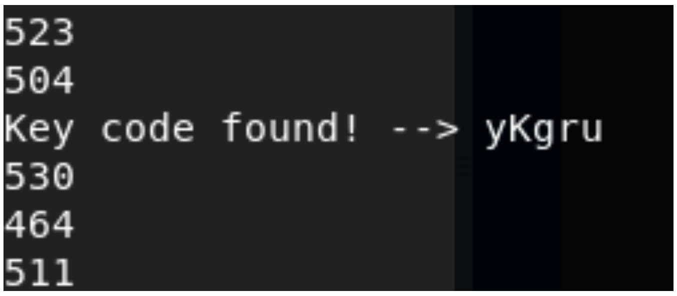
```c
int main(){
        int sum=0;
        int j=0;
        int i=0;


        while(i<1000){
                char* str1 = rand_string(6);
                //printf("%s\n",str1);
                for(j=0;j<10;j++){
                        sum += (int)str1[j];
                        //printf("%d-->%d\n",j,sum);
                }
                if(sum==530){
                        printf("Key code found! --> %s\n",str1);
                        //break;
                }
                printf("%d\n",sum);
                i++;
                sum=0;
        }
        return 0;
}
```

6. 
7. The driver program for it
8. Running the script



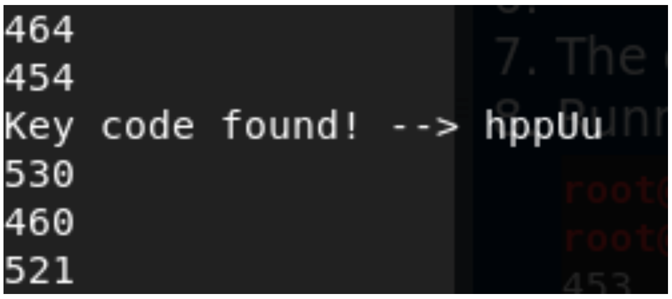
```
root@kali:~/liveoverflow# gcc -o scriptLog1 scriptLog1.c
root@kali:~/liveoverflow# ./scriptLog1
453
426
436
501
458
516
```

9.



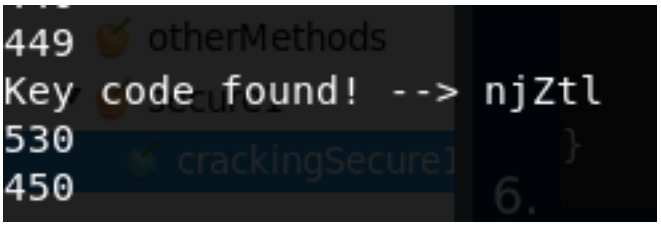
```
523
504
Key code found! --> yKgru
530
464
511
```

10.

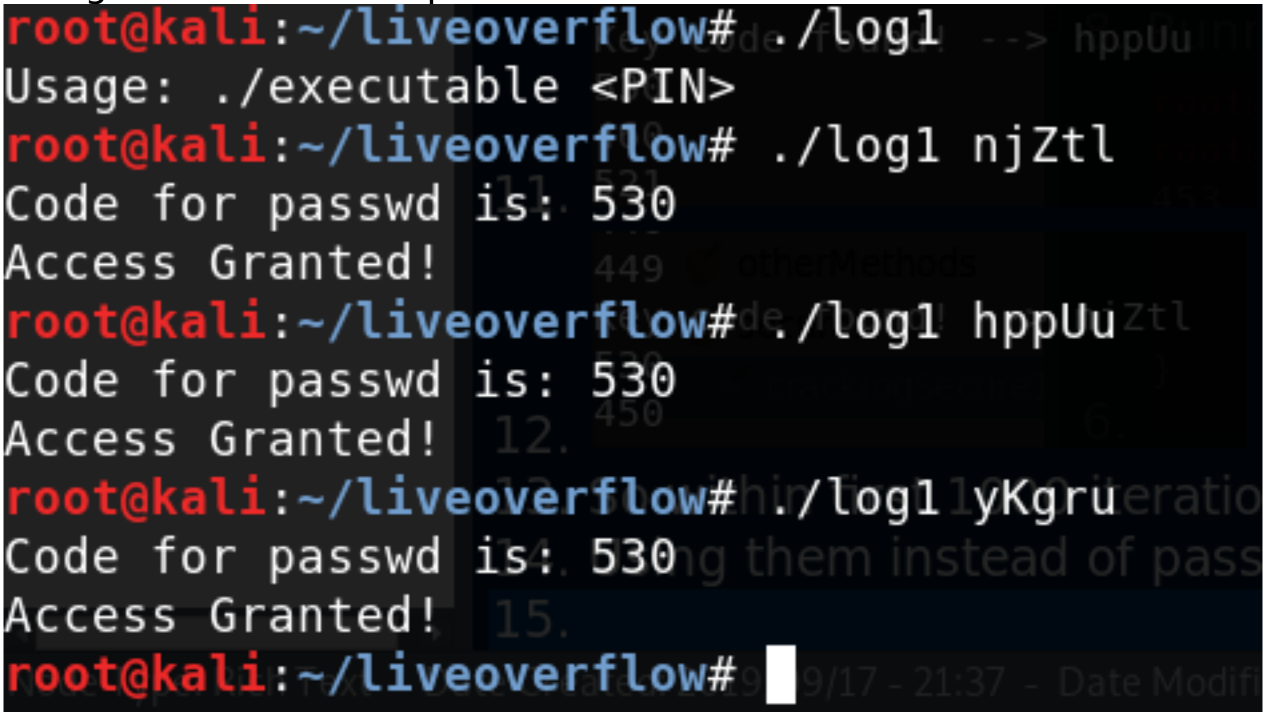


```
464
454
Key code found! --> hppUu
530
460
521
```

11.

12. 

```
449
Key code found! --> njZtl
530
450
```

13. So within first 1000 iterations we found nearly 4-5 keys
14. Using them instead of passwd
15. 

```
root@kali:~/liveoverflow# ./log1
Usage: ./executable <PIN>
root@kali:~/liveoverflow# ./log1 njZtl
Code for passwd is: 530
Access Granted!
root@kali:~/liveoverflow# ./log1 hppUu
Code for passwd is: 530
Access Granted!
root@kali:~/liveoverflow# ./log1 yKgru
Code for passwd is: 530
Access Granted!
root@kali:~/liveoverflow# 
```

16. Yeah Access granted!