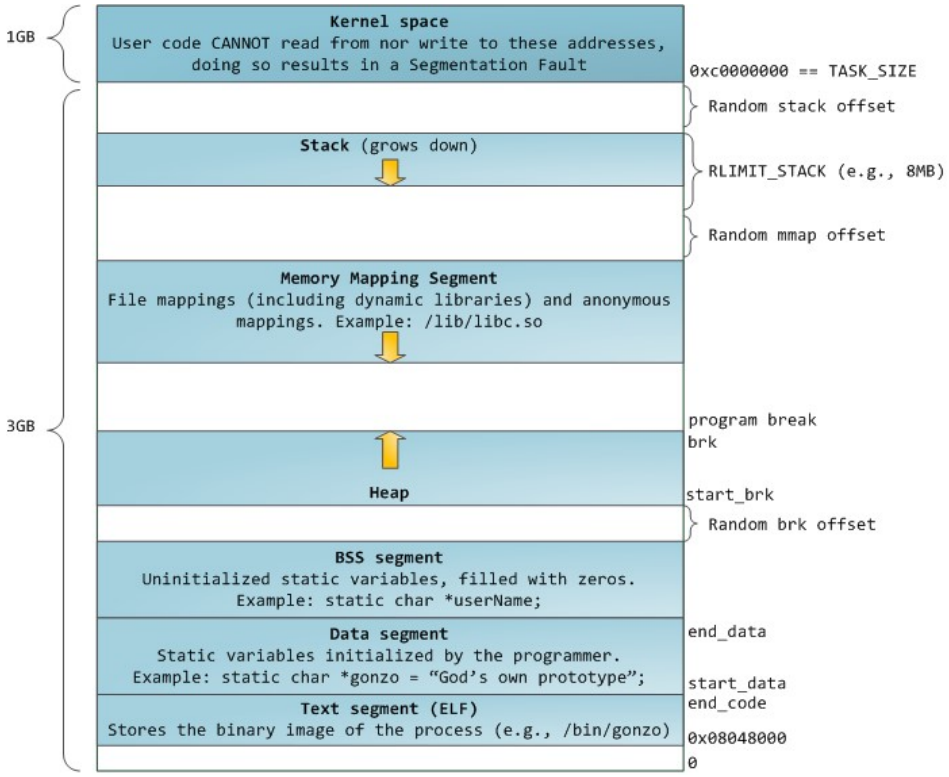
Syscalls used by malloc.

Posted on February 11, 2015 by sploitfun

Having landed on this page, you should know malloc uses syscalls to obtain memory from the OS. As shown in the below picture malloc invokes either brk or mmap syscall to obtain memory.

brk: brk obtains memory (non zero initialized) from kernel by increasing program break location (brk). Initially start (start_brk) and end of heap segment (brk) would point to same location.

- When ASLR is turned off, start_brk and brk would point to end of data/bss segment (end_data).
- When ASLR is turned on, start_brk and brk would be equal to end of data/bss segment (end_data) plus random brk offset.



Above "process virtual memory layout" picture shows start_brk is the beginning of heap segment and brk (program break) is the end of heap segment.

Example:

```c
/* sbrk and brk example */
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

```

```c
int main()
{
        void *curr_brk, *tmp_brk = NULL;

        printf("Welcome to sbrk example:%d\n", getpid());

        /* sbrk(0) gives current program break location */
        tmp_brk = curr_brk = sbrk(0);
        printf("Program Break Location1:%p\n", curr_brk);
        getchar();

        /* brk(addr) increments/decrements program break location */
        brk(curr_brk+4096);

        curr_brk = sbrk(0);
        printf("Program break Location2:%p\n", curr_brk);
        getchar();

        brk(tmp_brk);

        curr_brk = sbrk(0);
        printf("Program Break Location3:%p\n", curr_brk);
        getchar();

        return 0;
}
```

Output Analysis:

Before increasing program break: In the below output we can observe there is NO heap segment. Hence

- start_brk = brk = end_data = 0x804b000.

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ ./sbrk
Welcome to sbrk example:6141
Program Break Location1:0x804b000
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6141/maps
...
0804a000-0804b000 rw-p 00001000 08:01 539624     /home/sploitfun/ptmalloc.ppt/syscalls/sbrk
b7e21000-b7e22000 rw-p 00000000 00:00 0
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$
```

After increasing program break location: In the below output we can observe there is heap segment. Hence

- start_brk = end_data = 0x804b000
- brk = 0x804c000.

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ ./sbrk

Welcome to sbrk example:6141

Program Break Location1:0x804b000

Program Break Location2:0x804c000
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6141/maps
...
0804a000-0804b000 rw-p 00001000 08:01 539624
/home/sploitfun/ptmalloc.ppt/syscalls/sbrk
0804b000-0804c000 rw-p 00000000 00:00 0          [heap]
b7e21000-b7e22000 rw-p 00000000 00:00 0
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$
```

where
0804b000-0804c000 is Virtual address range for this segment
rw-p is Flags (Read, Write, NoeXecute, Private)
00000000 is File offset - Since its not mapped from any file, its zero here
00:00 is Major/Minor device number - Since its not mapped from any file, its zero here
0 is Inode number - Since its not mapped from any file, its zero here
[heap] is Heap segment

mmap: malloc uses mmap to create a private anonymous mapping segment. The primary purpose of private anonymous mapping is to allocate new memory (zero filled) and this new memory would be exclusively used by calling process.

Example:

```
/* Private anonymous mapping example using mmap syscall */
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

void static inline errExit(const char* msg)
{
        printf("%s failed. Exiting the process\n", msg);
        exit(-1);
}
```

```
int main()
{
        int ret = -1;
        printf("Welcome to private anonymous mapping example::PID:%d\n", getpid());
        printf("Before mmap\n");
        getchar();
        char* addr = NULL;
        addr = mmap(NULL, (size_t)132*1024, PROT_READ|PROT_WRITE, MAP_PRIVATE |
MAP_ANONYMOUS, -1, 0);
        if (addr == MAP_FAILED)
                errExit("mmap");
        printf("After mmap\n");
        getchar();

        /* Unmap mapped region. */
        ret = munmap(addr, (size_t)132*1024);
        if(ret == -1)
                errExit("munmap");
        printf("After munmap\n");
        getchar();
        return 0;
}
```

Output Analysis:

Before mmap: In the below output we can see only memory mapping segments that belongs to shared libraries libc.so and ld-linux.so

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6067/maps

08048000-08049000 r-xp 00000000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

08049000-0804a000 r--p 00000000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

0804a000-0804b000 rw-p 00001000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

b7e21000-b7e22000 rw-p 00000000 00:00 0
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$
```

After mmap: In the below output we can observe that our memory mapping segment (b7e00000 - b7e21000 whose size is 132KB) is combined with already existing memory mapping segment (b7e21000 - b7e22000).

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6067/maps
```

```
08048000-08049000 r-xp 00000000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

08049000-0804a000 r--p 00000000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

0804a000-0804b000 rw-p 00001000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

b7e00000-b7e22000 rw-p 00000000 00:00 0
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$
```

where
b7e00000-b7e22000 is Virtual address range for this segment
rw-p is Flags (Read, Write, NoeXecute, Private)
00000000 is File offset - Since its not mapped from any file, its zero here
00:00 is Major/Minor device number - Since its not mapped from any file, its zero here
0 is Inode number - Since its not mapped from any file, its zero here

After munmap: In the below output we can see that our memory mapping segment is unmapped ie) its corresponding memory is released to the operating system.

```
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$ cat /proc/6067/maps

08048000-08049000 r-xp 00000000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

08049000-0804a000 r--p 00000000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

0804a000-0804b000 rw-p 00001000 08:01 539691
/home/sploitfun/ptmalloc.ppt/syscalls/mmap

b7e21000-b7e22000 rw-p 00000000 00:00 0
...
sploitfun@sploitfun-VirtualBox:~/ptmalloc.ppt/syscalls$
```

NOTE: In our sample program executions ASLR was turned off.

Reference:

1. Anatomy of a program in memory