Hack 虚拟内存系列（三）：虚拟内存图解

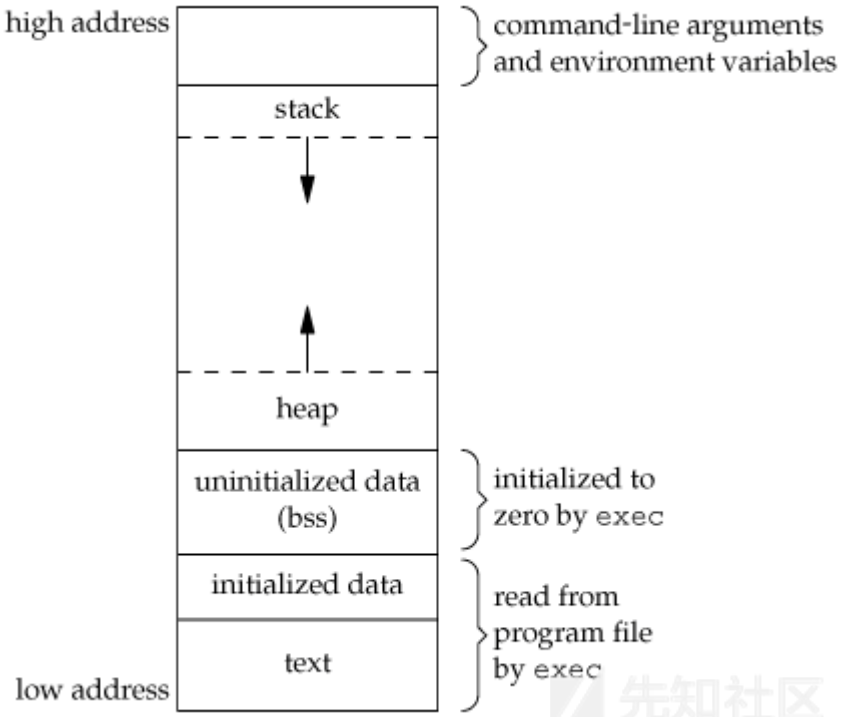
lm0963 / 2018-08-06 11:54:36 / 浏览数 2703 技术文章 技术文章 顶(0) 踩(0)

本文翻译自：Hack the Virtual Memory: drawing the VM diagram

Hack the Virtual Memory: drawing the VM diagram

Hack虚拟内存之第2章：虚拟内存图解

我们之前讨论过进程的虚拟内存中的内容。今天，我们将通过编写程序打印出各个元素的地址来"重建"（部分）下图。



前提

为了完全理解本文，你需要知道：

- C语言的基础知识
- 汇编的基础知识（非必需）
- Linux文件系统和shell的基础知识
- 我们将用到/proc/[pid]/maps文件（查阅man proc或阅读本系列的第一篇文章： 第0章：C字符串和/proc ）

环境

所有脚本和程序都已经在以下系统上进行过测试：

- Ubuntu 14.04 LTS
  - Linux ubuntu 4.4.0-31-generic #50~14.04.1-Ubuntu SMP Wed Jul 13 01:07:32 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
  - 下文描述均基于此系统，在其他系统上不一定成功

使用工具：

- gcc
  - gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
- objdump
  - GNU objdump (GNU Binutils for Ubuntu) 2.24
- udcli
  - udis86 1.7.2
- bc
  - bc 1.06.95

栈

我们想要在图中找到的第一个东西是栈。我们知道在C语言中，局部变量位于栈中。因此，如果我们打印局部变量的地址，那么我们就应该能知道栈在虚拟内存中的位置。让

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

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(void)
{
   int a;

   printf("Address of a: %p\n", (void *)&a);
   return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -pedantic -Werror main-0.c -o 0
julien@holberton:~/holberton/w/hackthevm2$ ./0
Address of a: 0x7ffd14b8bd9c
julien@holberton:~/holberton/w/hackthevm2$
```

当我们比较其他元素地址时，这将是我们的第一个参考点。

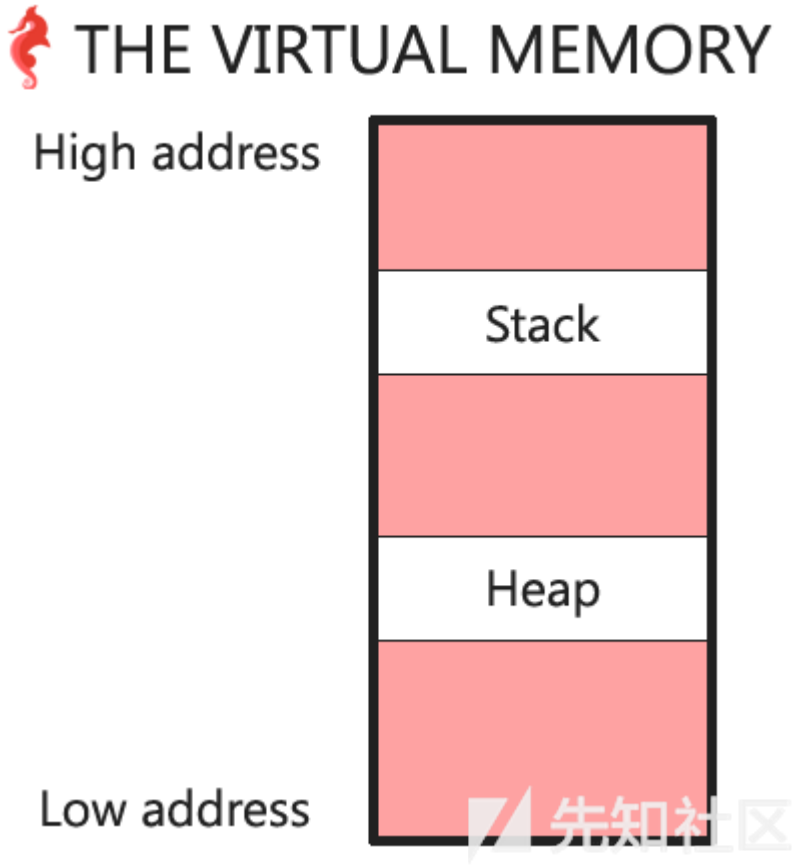
堆

当使用malloc为变量分配空间时，将会使用到堆。添加一行代码使用malloc分配空间并查看malloc返回的内存地址位置（main-1.c）：

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

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(void)
{
   int a;
   void *p;

   printf("Address of a: %p\n", (void *)&a);
   p = malloc(98);
   if (p == NULL)
   {
       fprintf(stderr, "Can't malloc\n");
       return (EXIT_FAILURE);
   }
   printf("Allocated space in the heap: %p\n", p);
   return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -pedantic -Werror main-1.c -o 1
julien@holberton:~/holberton/w/hackthevm2$ ./1
Address of a: 0x7ffd4204c554
Allocated space in the heap: 0x901010
julien@holberton:~/holberton/w/hackthevm2$
```

很明显堆地址（0x901010）远小于栈地址（0x7ffd4204c554）。此时我们可以画出草图（堆与栈）：



可执行代码区域

程序代码也在虚拟内存中。如果我们打印main函数的地址，我们可以知道程序代码与栈和堆的相对位置。看一下能否在堆下面找到程序代码（main-2.c）：

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

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(void)
{
   int a;
   void *p;

   printf("Address of a: %p\n", (void *)&a);
   p = malloc(98);
   if (p == NULL)
   {
       fprintf(stderr, "Can't malloc\n");
       return (EXIT_FAILURE);
   }
   printf("Allocated space in the heap: %p\n", p);
   printf("Address of function main: %p\n", (void *)main);
   return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -Werror main-2.c -o 2
julien@holberton:~/holberton/w/hackthevm2$ ./2
Address of a: 0x7ffdced37d74
Allocated space in the heap: 0x2199010
Address of function main: 0x40060d
julien@holberton:~/holberton/w/hackthevm2$
```

似乎我们的程序地址（0x40060d）正如预期般位于堆地址（0x2199010）下面。
但是，让我们确保这是我们程序的实际代码，而不是某种指向另一个位置的指针。用objdump反汇编我们的程序2并查看main函数的"内存地址"：

```
julien@holberton:~/holberton/w/hackthevm2$ objdump -M intel -j .text -d 2 | grep '<main>:' -A 5
000000000040060d <main>:
 40060d:   55                      push   rbp
 40060e:   48 89 e5                mov    rbp,rsp
 400611:   48 83 ec 10             sub    rsp,0x10
 400615:   48 8d 45 f4             lea    rax,[rbp-0xc]
 400619:   48 89 c6                mov    rsi,rax
```

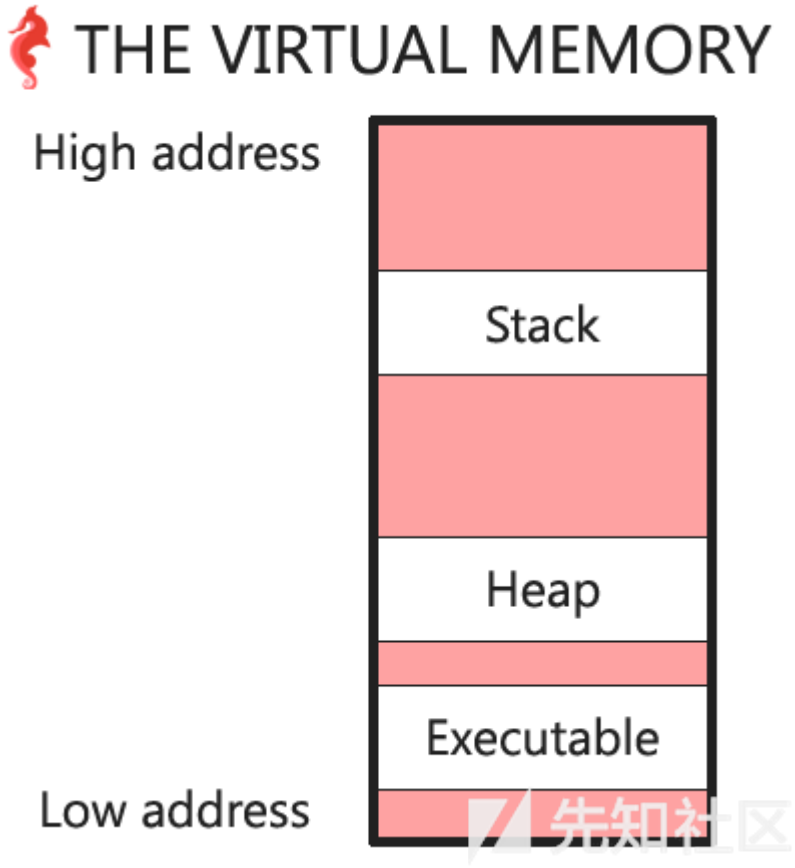
000000000040060d

<main> ->我们找到完全相同的地址（0x40060d）。如果仍不确定，可以打印位于此地址的第一个字节，以确保它们与objdump的输出匹配（main-3.c）：

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

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(void)
{
   int a;
   void *p;
   unsigned int i;

   printf("Address of a: %p\n", (void *)&a);
   p = malloc(98);
   if (p == NULL)
   {
       fprintf(stderr, "Can't malloc\n");
       return (EXIT_FAILURE);
   }
   printf("Allocated space in the heap: %p\n", p);
   printf("Address of function main: %p\n", (void *)main);
   printf("First bytes of the main function:\n\t");
   for (i = 0; i < 15; i++)
   {
       printf("%02x ", ((unsigned char *)main)[i]);
   }
   printf("\n");
   return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -Werror main-3.c -o 3
julien@holberton:~/holberton/w/hackthevm2$ objdump -M intel -j .text -d 3 | grep '<main>:' -A 5
000000000040064d <main>:
 40064d:   55                      push   rbp
 40064e:   48 89 e5                mov    rbp,rsp
 400651:   48 83 ec 10             sub    rsp,0x10
 400655:   48 8d 45 f0             lea    rax,[rbp-0x10]
 400659:   48 89 c6                mov    rsi,rax
julien@holberton:~/holberton/w/hackthevm2$ ./3
Address of a: 0x7ffeff0f13b0
Allocated space in the heap: 0x8b3010
Address of function main: 0x40064d
First bytes of the main function:
   55 48 89 e5 48 83 ec 10 48 8d 45 f0 48 89 c6
julien@holberton:~/holberton/w/hackthevm2$ echo "55 48 89 e5 48 83 ec 10 48 8d 45 f0 48 89 c6" | udcli -64 -x -o 40064d
000000000040064d 55              push rbp
000000000040064e 4889e5          mov rbp, rsp
0000000000400651 4883ec10        sub rsp, 0x10
0000000000400655 488d45f0        lea rax, [rbp-0x10]
0000000000400659 4889c6          mov rsi, rax
julien@holberton:~/holberton/w/hackthevm2$
```

->可以看到程序打印出（和objdump）相同的地址和相同的内容。现在可以十分确定这是main函数地址。
可以在此处下载Udis86反汇编程序库
以下是基于我们刚刚所了解的知识更新后的图表：



命令行参数和环境变量

main函数可以带参数：

- 命令行参数
  - main函数的第一个参数（通常命名为argc或ac）是命令行参数的数量
  - main函数的第二个参数（通常命名为argv或av）是一个指针数组指向输入的参数（C字符串）
- 环境变量
  - main函数的第三个参数（通常命名为env或envp）是一个指针数组指向环境变量（C字符串）

让我们看看这些元素在进程的虚拟内存中的位置（main-4.c）：

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

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(int ac, char **av, char **env)
{
	int a;
	void *p;
	int i;

	printf("Address of a: %p\n", (void *)&a);
	p = malloc(98);
	if (p == NULL)
	{
		fprintf(stderr, "Can't malloc\n");
		return (EXIT_FAILURE);
	}
	printf("Allocated space in the heap: %p\n", p);
```

```
        printf("Address of function main: %p\n", (void *)main);
        printf("First bytes of the main function:\n\t");
        for (i = 0; i < 15; i++)
        {
                printf("%02x ", ((unsigned char *)main)[i]);
        }
        printf("\n");
        printf("Address of the array of arguments: %p\n", (void *)av);
        printf("Addresses of the arguments:\n\t");
        for (i = 0; i < ac; i++)
        {
                printf("[%s]:%p ", av[i], av[i]);
        }
        printf("\n");
        printf("Address of the array of environment variables: %p\n", (void *)env);
    printf("Address of the first environment variable: %p\n", (void *)(env[0]));
        return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -Werror main-4.c -o 4
julien@holberton:~/holberton/w/hackthevm2$ ./4 Hello Holberton School!
Address of a: 0x7ffe7d6d8da0
Allocated space in the heap: 0xc8c010
Address of function main: 0x40069d
First bytes of the main function:
   55 48 89 e5 48 83 ec 30 89 7d ec 48 89 75 e0
Address of the array of arguments: 0x7ffe7d6d8e98
Addresses of the arguments:
   [./4]:0x7ffe7d6da373 [Hello]:0x7ffe7d6da377 [Holberton]:0x7ffe7d6da37d [School!]:0x7ffe7d6da387
Address of the array of environment variables: 0x7ffe7d6d8ec0
Address of the first environment variables:
   [0x7ffe7d6da38f]:"XDG_VTNR=7"
   [0x7ffe7d6da39a]:"XDG_SESSION_ID=c2"
   [0x7ffe7d6da3ac]:"CLUTTER_IM_MODULE=xim"
julien@holberton:~/holberton/w/hackthevm2$
```

命令行参数和环境变量地址在栈之上，现在我们知道了确切的顺序：stack（0x7ffe7d6d8da0）<argv（0x7ffe7d6d8e98）<env（0x7ffe7d6d8ec0）<arguments（从0x
实际上，可以看到所有命令行参数在内存中彼此相邻，也紧挨着环境变量。

argv和env数组是否彼此相邻？

argv数组有5个元素（命令行有4个元素，末尾有1个NULL元素（argv总是以NULL标记数组的结尾））。每个元素都是一个指向char型的指针，因为我们在64位机器上，所
* 8 =
40。10进制中的40转换成16进制是0x28。如果我们将这个值加到数组的起始地址（0x7ffe7d6d8e98），将会得到0x7ffe7d6d8ec0（env数组的起始地址）！所以两个数组

第一个命令行参数是否紧挨着存储在env数组之后？

为了确定这一点，我们需要知道env数组的大小。我们知道它以NULL指针结束，因此为了获得其元素的数量，我们只需要遍历它，检查"current"元素是否为NULL。这是更

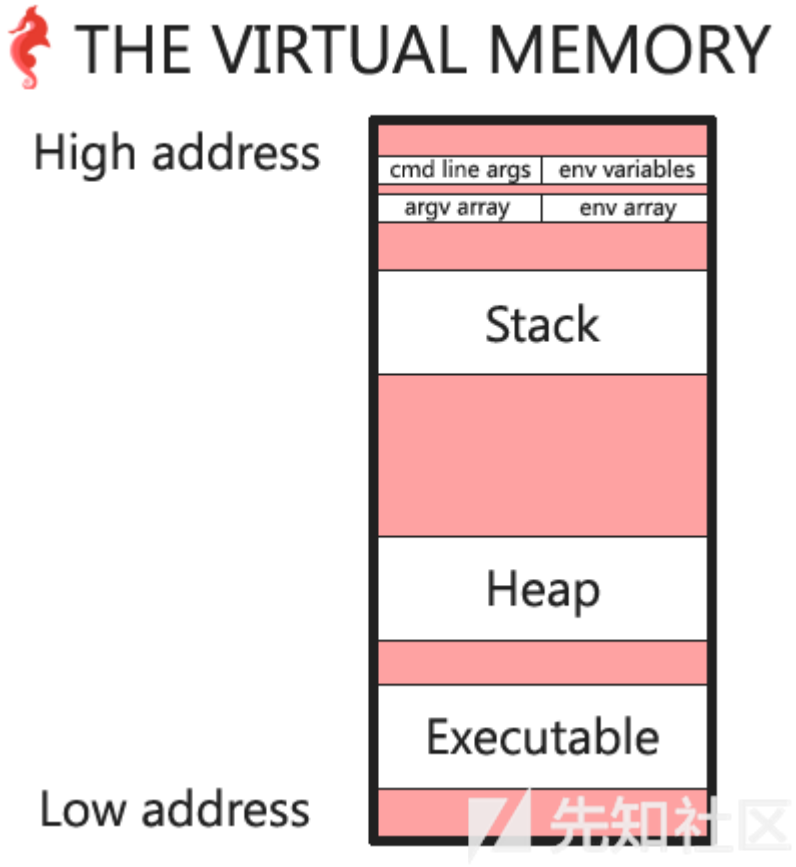
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(int ac, char **av, char **env)
{
    int a;
    void *p;
    int i;
    int size;

    printf("Address of a: %p\n", (void *)&a);
    p = malloc(98);
    if (p == NULL)
    {
         fprintf(stderr, "Can't malloc\n");
```

```
        return (EXIT_FAILURE);
    }
    printf("Allocated space in the heap: %p\n", p);
    printf("Address of function main: %p\n", (void *)main);
    printf("First bytes of the main function:\n\t");
    for (i = 0; i < 15; i++)
    {
        printf("%02x ", ((unsigned char *)main)[i]);
    }
    printf("\n");
    printf("Address of the array of arguments: %p\n", (void *)av);
    printf("Addresses of the arguments:\n\t");
    for (i = 0; i < ac; i++)
    {
        printf("[%s]:%p ", av[i], av[i]);
    }
    printf("\n");
    printf("Address of the array of environment variables: %p\n", (void *)env);
    printf("Address of the first environment variables:\n");
    for (i = 0; i < 3; i++)
    {
        printf("\t[%p]:\"%s\"\n", env[i], env[i]);
    }
    /* size of the env array */
    i = 0;
    while (env[i] != NULL)
    {
        i++;
    }
    i++; /* the NULL pointer */
    size = i * sizeof(char *);
    printf("Size of the array env: %d elements -> %d bytes (0x%x)\n", i, size, size);
    return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ ./5 Hello Betty Holberton!
Address of a: 0x7ffc77598acc
Allocated space in the heap: 0x2216010
Address of function main: 0x40069d
First bytes of the main function:
   55 48 89 e5 48 83 ec 40 89 7d dc 48 89 75 d0
Address of the array of arguments: 0x7ffc77598bc8
Addresses of the arguments:
   [./5]:0x7ffc7759a374 [Hello]:0x7ffc7759a378 [Betty]:0x7ffc7759a37e [Holberton!]:0x7ffc7759a384
Address of the array of environment variables: 0x7ffc77598bf0
Address of the first environment variables:
   [0x7ffc7759a38f]:"XDG_VTNR=7"
   [0x7ffc7759a39a]:"XDG_SESSION_ID=c2"
   [0x7ffc7759a3ac]:"CLUTTER_IM_MODULE=xim"
Size of the array env: 62 elements -> 496 bytes (0x1f0)
julien@holberton:~/holberton/w/hackthevm2$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
obase=16
ibase=16
1F0+7FFC77598BF0
7FFC77598DE0
quit
julien@holberton:~/holberton/w/hackthevm2$
```

->0x7FFC77598DE0！=（但仍小于）0x7ffc7759a374。所以答案是否定的

更新图表：



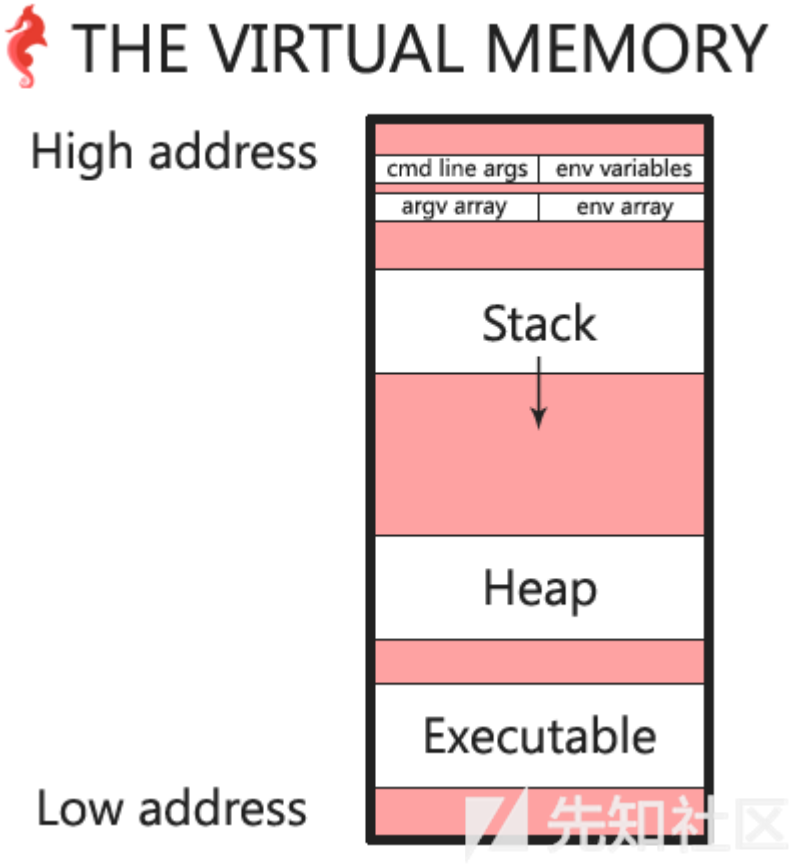
栈真的向下生长吗？

让我们调用一个函数并证实这点！如果是真的，则调用函数的变量的内存地址将大于被调用函数的变量的内存地址（main-6.c）。

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

/**
* f - print locations of various elements
*
* Returns: nothing
*/
void f(void)
{
    int a;
    int b;
    int c;

    a = 98;
    b = 1024;
    c = a * b;
    printf("[f] a = %d, b = %d, c = a * b = %d\n", a, b, c);
    printf("[f] Adresses of a: %p, b = %p, c = %p\n", (void *)&a, (void *)&b, (void *)&c);
}

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(int ac, char **av, char **env)
{
    int a;
    void *p;
    int i;
    int size;
```

```
    printf("Address of a: %p\n", (void *)&a);
    p = malloc(98);
    if (p == NULL)
    {
         fprintf(stderr, "Can't malloc\n");
         return (EXIT_FAILURE);
    }
    printf("Allocated space in the heap: %p\n", p);
    printf("Address of function main: %p\n", (void *)main);
    printf("First bytes of the main function:\n\t");
    for (i = 0; i < 15; i++)
    {
         printf("%02x ", ((unsigned char *)main)[i]);
    }
    printf("\n");
    printf("Address of the array of arguments: %p\n", (void *)av);
    printf("Addresses of the arguments:\n\t");
    for (i = 0; i < ac; i++)
    {
         printf("[%s]:%p ", av[i], av[i]);
    }
    printf("\n");
    printf("Address of the array of environment variables: %p\n", (void *)env);
    printf("Address of the first environment variables:\n");
    for (i = 0; i < 3; i++)
    {
         printf("\t[%p]:\"%s\"\n", env[i], env[i]);
    }
    /* size of the env array */
    i = 0;
    while (env[i] != NULL)
    {
         i++;
    }
    i++; /* the NULL pointer */
    size = i * sizeof(char *);
    printf("Size of the array env: %d elements -> %d bytes (0x%x)\n", i, size, size);
    f();
    return (EXIT_SUCCESS);
}

julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -Werror main-6.c -o 6
julien@holberton:~/holberton/w/hackthevm2$ ./6
Address of a: 0x7ffdae53ea4c
Allocated space in the heap: 0xf32010
Address of function main: 0x4006f9
First bytes of the main function:
   55 48 89 e5 48 83 ec 40 89 7d dc 48 89 75 d0
Address of the array of arguments: 0x7ffdae53eb48
Addresses of the arguments:
   [./6]:0x7ffdae54038b
Address of the array of environment variables: 0x7ffdae53eb58
Address of the first environment variables:
   [0x7ffdae54038f]:"XDG_VTNR=7"
   [0x7ffdae54039a]:"XDG_SESSION_ID=c2"
   [0x7ffdae5403ac]:"CLUTTER_IM_MODULE=xim"
Size of the array env: 62 elements -> 496 bytes (0x1f0)
[f] a = 98, b = 1024, c = a * b = 100352
[f] Adresses of a: 0x7ffdae53ea04, b = 0x7ffdae53ea08, c = 0x7ffdae53ea0c
julien@holberton:~/holberton/w/hackthevm2$
```

->是真的！（函数f中var a的地址）0x7ffdae53ea04 <0x7ffdae53ea4c（函数main中var a的地址）
现在更新我们的图表：



/proc

让我们用/proc/[pid]/maps（man proc或参考本系列的第一篇文章，了解proc文件系统，如果你不知道它是什么）来仔细检查我们到目前为止找到的所有内容。让我们在程序中添加一个getchar（）语句，以便我们可以查看它的"/proc"（main-7.c）：

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

/**
* f - print locations of various elements
*
* Returns: nothing
*/
void f(void)
{
    int a;
    int b;
    int c;

    a = 98;
    b = 1024;
    c = a * b;
    printf("[f] a = %d, b = %d, c = a * b = %d\n", a, b, c);
    printf("[f] Adresses of a: %p, b = %p, c = %p\n", (void *)&a, (void *)&b, (void *)&c);
}

/**
* main - print locations of various elements
*
* Return: EXIT_FAILURE if something failed. Otherwise EXIT_SUCCESS
*/
int main(int ac, char **av, char **env)
{
    int a;
    void *p;
    int i;
    int size;
```

```
    printf("Address of a: %p\n", (void *)&a);
    p = malloc(98);
    if (p == NULL)
    {
         fprintf(stderr, "Can't malloc\n");
         return (EXIT_FAILURE);
    }
    printf("Allocated space in the heap: %p\n", p);
    printf("Address of function main: %p\n", (void *)main);
    printf("First bytes of the main function:\n\t");
    for (i = 0; i < 15; i++)
    {
         printf("%02x ", ((unsigned char *)main)[i]);
    }
    printf("\n");
    printf("Address of the array of arguments: %p\n", (void *)av);
    printf("Addresses of the arguments:\n\t");
    for (i = 0; i < ac; i++)
    {
         printf("[%s]:%p ", av[i], av[i]);
    }
    printf("\n");
    printf("Address of the array of environment variables: %p\n", (void *)env);
    printf("Address of the first environment variables:\n");
    for (i = 0; i < 3; i++)
    {
         printf("\t[%p]:\"%s\"\n", env[i], env[i]);
    }
    /* size of the env array */
    i = 0;
    while (env[i] != NULL)
    {
         i++;
    }
    i++; /* the NULL pointer */
    size = i * sizeof(char *);
    printf("Size of the array env: %d elements -> %d bytes (0x%x)\n", i, size, size);
    f();
    getchar();
    return (EXIT_SUCCESS);
}
```

```
julien@holberton:~/holberton/w/hackthevm2$ gcc -Wall -Wextra -Werror main-7.c -o 7
julien@holberton:~/holberton/w/hackthevm2$ ./7 Rona is a Legend SRE
Address of a: 0x7fff16c8146c
Allocated space in the heap: 0x2050010
Address of function main: 0x400739
First bytes of the main function:
   55 48 89 e5 48 83 ec 40 89 7d dc 48 89 75 d0
Address of the array of arguments: 0x7fff16c81568
Addresses of the arguments:
   [./7]:0x7fff16c82376 [Rona]:0x7fff16c8237a [is]:0x7fff16c8237f [a]:0x7fff16c82382 [Legend]:0x7fff16c82384 [SRE]:0x7fff16c82
Address of the array of environment variables: 0x7fff16c815a0
Address of the first environment variables:
   [0x7fff16c8238f]:"XDG_VTNR=7"
   [0x7fff16c8239a]:"XDG_SESSION_ID=c2"
   [0x7fff16c823ac]:"CLUTTER_IM_MODULE=xim"
Size of the array env: 62 elements -> 496 bytes (0x1f0)
[f] a = 98, b = 1024, c = a * b = 100352
[f] Adresses of a: 0x7fff16c81424, b = 0x7fff16c81428, c = 0x7fff16c8142c

julien@holberton:~$ ps aux | grep "./7" | grep -v grep
julien     5788  0.0  0.0   4336   628 pts/8    S+   18:04   0:00 ./7 Rona is a Legend SRE
julien@holberton:~$ cat /proc/5788/maps
00400000-00401000 r-xp 00000000 08:01 171828                             /home/julien/holberton/w/hackthevm2/7
00600000-00601000 r--p 00000000 08:01 171828                             /home/julien/holberton/w/hackthevm2/7
00601000-00602000 rw-p 00001000 08:01 171828                             /home/julien/holberton/w/hackthevm2/7
02050000-02071000 rw-p 00000000 00:00 0                                  [heap]
```

```
7f68caa1c000-7f68cabd6000 r-xp 00000000 08:01 136253                     /lib/x86_64-linux-gnu/libc-2.19.so
7f68cabd6000-7f68cadd6000 ---p 001ba000 08:01 136253                     /lib/x86_64-linux-gnu/libc-2.19.so
7f68cadd6000-7f68cadda000 r--p 001ba000 08:01 136253                     /lib/x86_64-linux-gnu/libc-2.19.so
7f68cadda000-7f68caddc000 rw-p 001be000 08:01 136253                     /lib/x86_64-linux-gnu/libc-2.19.so
7f68caddc000-7f68cade1000 rw-p 00000000 00:00 0
7f68cade1000-7f68cae04000 r-xp 00000000 08:01 136229                     /lib/x86_64-linux-gnu/ld-2.19.so
7f68cafe8000-7f68cafeb000 rw-p 00000000 00:00 0
7f68cafff000-7f68cb003000 rw-p 00000000 00:00 0
7f68cb003000-7f68cb004000 r--p 00022000 08:01 136229                     /lib/x86_64-linux-gnu/ld-2.19.so
7f68cb004000-7f68cb005000 rw-p 00023000 08:01 136229                     /lib/x86_64-linux-gnu/ld-2.19.so
7f68cb005000-7f68cb006000 rw-p 00000000 00:00 0
7fff16c62000-7fff16c83000 rw-p 00000000 00:00 0                          [stack]
7fff16d07000-7fff16d09000 r--p 00000000 00:00 0                          [vvar]
7fff16d09000-7fff16d0b000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
julien@holberton:~$
```

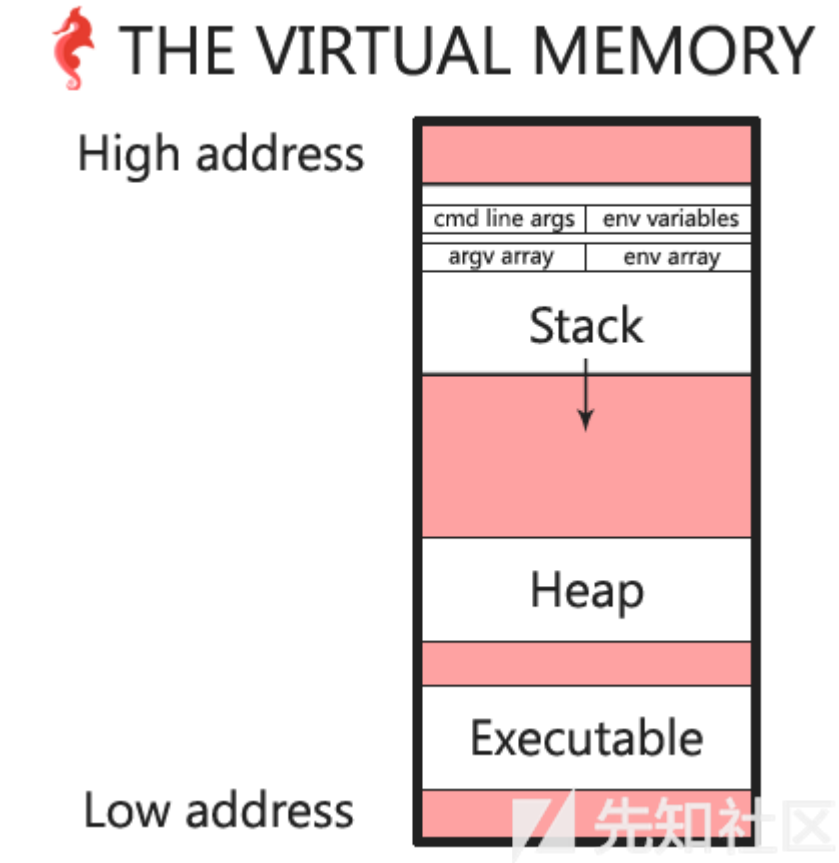
让我们来确定几件事：

- 栈起始于0x7fff16c62000，结束于0x7fff16c83000。我们的变量都在这个区域内（0x7fff16c8146c，0x7fff16c81424,0x7fff16c81428,0x7fff16c8142c）
- 堆起始于0x02050000，结束于0x02071000。我们分配的内存在此位置（0x2050010）
- 我们的代码（主函数）位于地址0x400739处，在以下区域内：
  00400000-00401000 r-xp 00000000 08:01 171828 /home/julien/holberton/w/hackthevm2/7
  它来自于文件/home/julien/holberton/w/hackthevm2/7（我们的可执行文件），并且该区域具有执行权限，这是合理的。
- 参数和环境变量（从0x7fff16c81568到0x7fff16c8238f + 0x1f0）位于从0x7fff16c62000开始到0x7fff16c83000的区域，，，栈！所以他们在栈中，而不是在栈之外。

这也带来了更多问题：

- 为什么我们的可执行文件"划分"为三个具有不同权限的区域？在下面这两个区域内有什么？
  - 00600000-00601000 r--p 00000000 08:01 171828 /home/julien/holberton/w/hackthevm2/7
  - 00601000-00602000 rw-p 00001000 08:01 171828 /home/julien/holberton/w/hackthevm2/7
- 其他那些区域是什么？
- 为什么我们分配的内存不在堆的最开始处（0x2050010 vs 02050000）？前16个字节被用来干什么？

还有一件事我们没有检查过：堆真的在向上增长吗？
以后我们会知道的！在我们结束本章之前，让我们用我们学到的所有东西来更新我们的图表：



</main>

上一篇：Pwn2Own 2018 Safa... 下一篇：Hack 虚拟内存系列（四）：ma...
1. 0 条回复
   - 动动手指，沙发就是你的了！