#**How To Create Your Own Shellcode On Arch Linux ?**
#**Author : N3td3v!l**
#**Contact-mail :** 4nonymouse@usa.com
#**Website : Nopotm.ir**
#**Spcial tnx to : C0nn3ct0r And All Honest Hackerz and Security Managers**

I - Presentation Of The Shellcode

A shellcode is a string that represents an executable binary code capable of launching any application on the machine. Most of the time shellcode opens a shell to have full access to the machine. Generally, the shellcode is injected into the memory of the machine via the exploitation of a buffer overflow flaw type (buffer overflow).

II - Understanding the basics

1 - System calls

A system call (in English, system call, syscall abbreviated) is a function provided by the kernel an operating system. According to our syscall Os will be called differently.Example of commonly used system call:

open, **write**, **read**, **close**, **chmod**, **chown** ...

On most operating systems, system calls can be used as simple functions written in C. For example for the chown system call:

extern int chown (__const char *__file, __uid_t __owner, __gid_t __group)

Each system call to an address that is assigned by the operating system and which is own. For example Linux kernel 2.6.31 with the address of syscall chown is 0xb6.
How do I know this address? A simple command like below allows for address the syscall. The addresses are in decimal unistd_x.h.

For a 32-bit

```
N3td3v!l@archlinux [ shellcode ]# cat /usr/include/asm/unistd_32.h | grep chown
    #define __NR_lchown          16
    #define __NR_fchown          95
    #define __NR_chown          182
    #define __NR_lchown32        198
    #define __NR_fchown32        207
    #define __NR_chown32         212
    #define __NR_fchownat        298
```

For a 64-bit system

```
N3td3v!l@archlinux [ shellcode ]# cat /usr/include/asm/unistd_64.h | grep chown
#define __NR_chown                      92
__SYSCALL(__NR_chown, sys_chown)
#define __NR_fchown                     93
__SYSCALL(__NR_fchown, sys_fchown)
#define __NR_lchown                     94
__SYSCALL(__NR_lchown, sys_lchown)
#define __NR_fchownat                   260
__SYSCALL(__NR_fchownat, sys_fchownat)
```

As you can see, if the bone is under 32 or 64 bits, the address of syscalls change.

III - Write the first shellcode

First we will create a simple shellcode, which will allow us to pause. Why we call the function whose address is _pause 29 resulting in hexadecimal 0x1d (in 32 bits).

```
N3td3v!l@archlinux [ ~ ]$ cat /usr/include/asm/unistd_32.h | grep
pause       #define __NR_pause              29
```

Once you know the address of the syscall, it remains for us to know what to put on the books. For this refer to this page =>http://www.potomacparent.com/cache/syscalls.html
We can see that to break we do not need to complete records, just a call away, which is going to be very short schedule.

```
N3td3v!l@archlinux [ shellcode ]$ cat pause.s
xor %eax,%eax
mov $29,%al
int $0x80
N3td3v!l@archlinux [ shellcode ]$ as -o pause.o pause.s
N3td3v!l@archlinux [ shellcode ]$ ld -o pause pause.o
ld: warning: cannot find entry symbol _start; defaulting to 08048054
N3td3v!l@archlinux [ shellcode ]$ ./pause
^C
N3td3v!l@archlinux [ shellcode ]$
```

Explanation :
#xor %eax,%eax <= We put the register eax to 0 to avoid segmentation faults
#mov $29,%al <= Placed 29 (the address of the syscall) in the register al
#int $0x80 <= executed

Now we will write C For this we need to know the value of work in hexadecimal asm what will eventually be our shellcode.
How have the hexadecimal equivalent?
It's simple, we simply use the objdump tool, which gives:

```
N3td3v!l@archlinux [ shellcode ]$ objdump -d ./pause
pause:     file format elf32-i386
Disassembly of section .text:
    08048054 <.text>:
    8048054:      31 c0                xor    %eax,%eax
    8048056:      b0 1d                mov    $0x1d,%al
    8048058:      cd 80                int    $0x80
N3td3v!l@archlinux [ shellcode ]$
```

And now, so the code is in C:

```
N3td3v!l@archlinux [ shellcode ]$ cat pause_c.c
    #include <stdio.h>

    void main(void)
    {
    char shellcode[] = "\x31\xc0\xb0\x1d\xcd\x80";

    (*(void(*)()) shellcode)();

    }
N3td3v!l@archlinux [ shellcode ]$ gcc -o pause_c pause_c.c
N3td3v!l@archlinux [ shellcode ]$ ./pause_c
^C
N3td3v!l@archlinux [ shellcode ]$
```

Your first shellcode working properly.
Now we will study the write function. We still refer to the site that I submitted earlier.

Info Register:

#%eax = 4
#%ebx = unsigned int
#%ecx = const char *
#%edx = size

We will simply write N3td3v!l, look at what gives the sources:

```
N3td3v!l@ArchLinux [shellcode]$ cat write.s
    ;_write
    xor   %eax,%eax   <= To avoid segmentation faults
    xor   %ebx,%ebx   <=   //          //
    xor   %ecx,%ecx   <=   //          //
    xor   %edx,%edx   <=   //          //

    movb  $0x9,%dl    <= placed the size of our word in dl (edx) so N3td3v! l + \ n l = 1 8 9
    pushl $0x0a       <= we begin to stack our line feed (\ n) = 0x0a
    push  $0x6e616874 <= seam
    push  $0x616e6f6a <= //          //
    movl  %esp,%ecx   <= % esp is sent to% ecx register that contains the constant tank _write
    movb  $0x1,%bl    <= here for 1% ebx,
    movb  $0x4,%al    <= and by the syscall so _write 4
    int   $0x80       <= executed

    ;_exit
    xor   %ebx,%ebx   <= %ebx = 0
    movb  $0x1,%al    <= %eax = 1 (_exit syscall)
    int   $0x80       <= executed
```

Compile and run our program:

```
N3td3v!l@ArchLinux [shellcode]$ as -o write.o write.s
N3td3v!l@ArchLinux [shellcode]$ ld -o write write.o
ld: warning: cannot find entry symbol _start; defaulting to 08048054
N3td3v!l@ArchLinux [shellcode]$ ./write
N3td3v!l
N3td3v!l@ArchLinux [shellcode]$
```

Let's write our shellcode in C for this, objdump will help a little.

```
N3td3v!l@ArchLinux [shellcode]$ objdump -d write
 write:     file format elf32-i386
    Disassembly of section .text:

     08048054 <.text>:
     8048054:       31 c0                   xor    %eax,%eax
     8048056:       31 db                   xor    %ebx,%ebx
     8048058:       31 c9                   xor    %ecx,%ecx
     804805a:       31 d2                   xor    %edx,%edx
     804805c:       b2 09                   mov    $0x9,%dl
     804805e:       6a 0a                   push   $0xa
     8048060:       68 74 68 61 6e          push   $0x6e616874
     8048065:       68 6a 6f 6e 61          push   $0x616e6f6a
     804806a:       89 e1                   mov    %esp,%ecx
     804806c:       b3 01                   mov    $0x1,%bl
     804806e:       b0 04                   mov    $0x4,%al
     8048070:       cd 80                   int    $0x80
     8048072:       31 db                   xor    %ebx,%ebx
     8048074:       b0 01                   mov    $0x1,%al
     8048076:       cd 80                   int    $0x80
N3td3v!l@ArchLinux [shellcode]$
```

There are many sources right in our asm code and then the equivalence of instructions in hexadecimal.

```
N3td3v!l@ArchLinux [shellcode]$ cat write_c.c
    #include <stdio.h>

    void main(void)
    {
    char shellcode[] =      "\x31\xc0\x31\xdb\x31\xc9"
                            "\x31\xd2\xb2\x09\x6a\x0a"
                            "\x68\x74\x68\x61\x6e\x68"
                            "\x6a\x6f\x6e\x61\x89\xe1"
                            "\xb3\x01\xb0\x04\xcd\x80"
                            "\x31\xdb\xb0\x01\xcd\x80";

    fprintf(stdout,"Lenght: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    }
```

Compile and execute our shellcode:

```
N3td3v!l@ArchLinux [shellcode]$ gcc -o write_c write_c.c
N3td3v!l@ArchLinux [shellcode]$ ./write_c
Lenght: 36
N3td3v!l
N3td3v!l@ArchLinux [shellcode]$
```

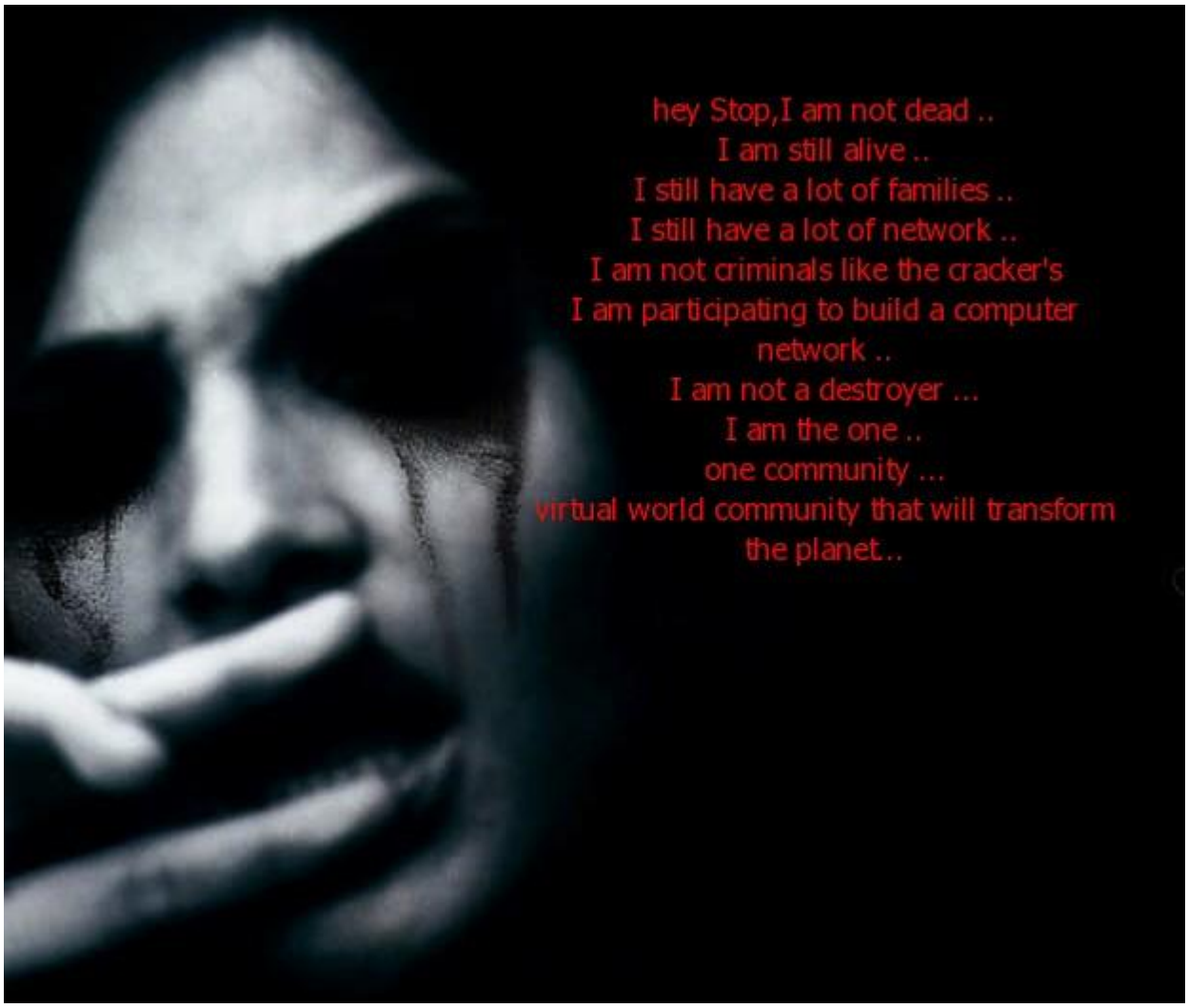
And here it works perfectly. Shellcode _write (1, "N3td3v!l S \ n", 9) + _exit (0) to a size of 36 bytes.

IV - References

[x] - http://wikipedia.org/wiki/Shellcode

[1] - /usr/include/asm/unistd_32.h

[2] – and etc...



========== Signature ==========