OpenBSD Remote Exploit

"Only two remote holes in the default install"

Alfredo A. Ortega

June 30, 2007
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.

The function `m_freem()` crashed...
Searching for a way to gain code execution
Searching for a way to gain code execution
C code equivalent

```c
#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
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)
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
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
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
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:

```asm
sl dt ax  ; Store LDT index on EAX
sub esp, byte 0x7f
sg dt [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, 0xffff0000
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)**

1. mprotect()
2. fork()
3. Standard user-mode code

**Creating a $W+X$ section**

1. fork()
2. mmap()
3. copy
4. jmp to mmaped

mprotect() extends CS permanently

**Figure:** Payload injection options
Questions before going on?

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