

相关资源

PNG文件格式文档

<http://www.libpng.org/pub/png/spec/1.2/PNG-Chunks.html>
<https://www.myway5.com/index.php/2017/11/10/png%E6%A0%BC%E5%BC%8F%E5%88%86%E6%9E%90%E4%B8%8E%E5%8E%8B%E7%BC%A9%E5%8E%9F%E7%90%>

源码下载

<http://78.108.103.11/MIRROR/ftp/png/src/history/libpng12/>

测试样本

<https://gitee.com/hac425/data/tree/master/libpng>

CVE-2004-0597

分析

漏洞代码

```
void /* PRIVATE */
png_handle_tRNS(png_structp png_ptr, png_infop info_ptr, png_uint_32 length)
{
    png_byte readbuf[PNG_MAX_PALETTE_LENGTH]; // 0x100
    .....
    .....
    png_crc_read(png_ptr, readbuf, (png_size_t)length);
    png_ptr->num_trans = (png_uint_16)length;
```

readbuf 是一个 0x100字节的缓冲区，length从 png 文件中读取，最大可以为 0x7ffffff，典型的栈溢出。

测试用例:

trns_stack_bof.png

Template Results - PNGTemplate.bt						
Name	Value	Start	Size	Color	Comment	
uint64 pngid	89504E470D0A1A0Ah	0h	8h	Fg: Bg:		
> struct CHUNK chunk[0]	IHDR (Critical, Public, Unsafe to Copy)	8h	19h	Fg: Bg:		
▼ struct CHUNK chunk[1]	tRNS (Ancillary, Public, Unsafe to Copy)	21h	20Ch	Fg: Bg:		
uint32 length	200h	21h	4h	Fg: Bg:		
> union CTYPE type	tRNS	25h	4h	Fg: Bg:		
> ubyte data[512]		29h	200h	Fg: Bg:		
uint32 crc	DD403A8Ah	229h	4h	Fg: Bg:		



修复

对 length进行校验避免大于 PNG_MAX_PALETTE_LENGTH.

[CVE-2007-5266](#)

补丁地址

<https://sourceforge.net/p/png-mng/mailman/png-mng-implement/thread/5122753600C3E94F87FBDFCC090D1FF0400EA68@MERCMBX07.na.sas.c>

iCCP 的格式

iccp_name■■■■+"\x00" + "\x00" + zlib■■■■■■■■ endata

endata 解压后的格式

profile_size 4

iCCP chunk 处理堆越界（基本无影响）

分析

调试环境

ubuntu 16.04 64bit

测试用例

libpng\iccp_memleak*.png

漏洞代码

```
#if defined(PNG_iCCP_SUPPORTED)
void PNGAPI
png_set_iCCP(png_structp png_ptr, png_infop info_ptr,
             png_charp name, int compression_type,
             png_charp profile, png_uint_32 proflen)
{
    new_iccp_name = (png_charp)png_malloc_warn(png_ptr, png_strlen(name)+1);

    png_strncpy(new_iccp_name, name, png_sizeof(new_iccp_name)); // name 8 strncpy 0
```

strncpy 的工作为

```
char* strncpy(char *dest, const char *src, size_t n){
    size_t i;
    for (i = 0 ; i < n && src[i] != '\0' ; i++)
        dest[i] = src[i];
    for ( ; i < n ; i++)
        dest[i] = '\0';
    return dest;
}
```

1. 如果src的前n个字符里面没有'\0'，那么它不会在末尾补上这个结束符
2. 如果拷贝的数据不满n个字符，那么它会用 '\0' 在末尾填充

漏洞是存在的，无影响的原因是在 linux x64 下分配内存的最小数据块为 0x20，可用的数据区域为 0x10。而 png_sizeof(new_iccp_name) 的大小为 8，所以不会溢出到其他内存块的数据里面。

Case1

当 iccp_name 的长度大于 8 时，strncpy 不会再字符串末尾填 \x00，后面的使用可能会导致内存数据泄露（比如分配到的内存块是位于 unsorted bin 中）。

iccp_memleak1.png

泄露堆块的指针

```
694 new_iccp_profile = (png_charp)png_malloc_warn(png_ptr, proflen);
gef> p new_iccp_name
$2 = (png_charp) 0x60ce50 "kkkkkkkkx;\272\367\377\177"
gef> hexdump 0x60ce50
[!] Syntax
hexdump (qword|dword|word|byte) ADDRESS [[L][SIZE]] [UP|DOWN] [S]
gef> hexdump byte 0x60ce50
0x00000000000060ce50  6b 6b 6b 6b 6b 6b 6b 6b 78 3b ba f7 ff 7f 00 00  kkkkkkkkx;.....
0x00000000000060ce60  90 e6 60 00 00 00 00 00 31 00 00 00 00 00 00 00  ..`.....1.....
0x00000000000060ce70  78 3b ba f7 ff 7f 00 00 78 3b ba f7 ff 7f 00 00  x;.....x;.....
0x00000000000060ce80  f0 d2 60 00 00 00 00 00 78 3b ba f7 ff 7f 00 00  ..`.....x;.....
gef>
```

先知社区

Case2

当 iccp_name 的长度小于 8 时，malloc 的大小会小于 png_sizeof(new_iccp_name)，这个会造成越界。

iccp_memleak2.png

当 iccp_name 为 k\x00 时，分配 2 字节

```
->0x7ffff7bb3332 <png_set_iCCP+101> call 0x7ffff7baf30 <png_malloc_warn@plt>
\-> 0x7ffff7baf30 <png_malloc_warn@plt+0> jmp QWORD PTR [rip+0x2262d2] # 0x7ffff7dd6108
0x7ffff7baf36 <png_malloc_warn@plt+6> push 0x1e
0x7ffff7baf3b <png_malloc_warn@plt+11> jmp 0x7ffff7bafc40
0x7ffff7baf40 <png_push_process_row@plt+0> jmp QWORD PTR [rip+0x2262ca] # 0x7ffff7dd6110
0x7ffff7baf46 <png_push_process_row@plt+6> push 0x1f
0x7ffff7baf4b <png_push_process_row@plt+11> jmp 0x7ffff7bafc40
-----
png_malloc_warn@plt (
    $rdi = 0x0000000000006880->0x00007ffff7bce5d0->0x20676e7062696c20,
    $rsi = 0x0000000000000002,
    $rdx = 0x0000000000000002,
    $rcx = 0x0000000000000650
)
-----
682
683 png_debug1(1, "in %s storage function\n", "iCCP");
684 if (png_ptr == NULL || info_ptr == NULL || name == NULL || profile == NULL)
685     return;
686
-> 687 new_iccp_name = (png_charp)png_malloc_warn(png_ptr, png_strlen(name)+1);
688 if (new_iccp_name == NULL)
```

后面拷贝时会拷贝 8 字节

```
0x7ffff7bafd7b <png_write_PLTE@plt+11> jmp 0x7ffff7bafc40
-----
strncpy@plt (
    $rdi = 0x0000000000006e63->0x0000000000000000,
    $rsi = 0x0000000000006e65->0x000500000000006b ("k?"),
    $rdx = 0x0000000000000008,
    $rcx = 0x0000000000006e65->0x000500000000006b ("k?")
)
-----
688 if (new_iccp_name == NULL)
689 {
690     png_warning(png_ptr, "Insufficient memory to process iCCP chunk.");
691     return;
692 }
-> 693 png_strncpy(new_iccp_name, name, png_sizeof(new_iccp_name));
694 new_iccp_profile = (png_charp)png_malloc_warn(png_ptr, proflen);
695 if (new_iccp_profile == NULL)
```

修复方式

```
png_strncpy(new_iccp_name, name, png_strlen(new_iccp_name)+1);
```

总结：strncpy 要小心使用

sPLT Chunk处理

分析

测试用例

```
■■■\libpng\splt.png
```

sPLT 的数据域的格式

```
■■■ + '\x00' + entries
```

entries 的结构

```
depth■ ■■■■■■entry■size: 1■■
entry ■■
```

entry 的结构

```
typedef struct png_sPLT_entry_struct
{
    png_uint_16 red;
    png_uint_16 green;
    png_uint_16 blue;
    png_uint_16 alpha;
    png_uint_16 frequency;
} png_sPLT_entry;
```

还是 strncpy 的使用，没有设置 '\x00' 可能会 leak.

```
png_set_sPLT(png_structp png_ptr,
             png_infop info_ptr, png_sPLT_tp entries, int nentries)
{
to->name = (png_charp)png_malloc_warn(png_ptr, png_strlen(from->name) + 1);
png_strncpy(to->name, from->name, png_strlen(from->name));
```

假设 from->name 为 8 字节

```
981     }
982     /* TODO: use png_malloc_warn */
983     png_strncpy(to->name, from->name, png_strlen(from->name));
984     to->entries = (png_sPLT_entryp)png_malloc_warn(png_ptr,
985 // nentries=0x1L
986             from->nentries * png_sizeof(png_sPLT_entry));
987     /* TODO: use png_malloc_warn */
988     png_memcpy(to->entries, from->entries,
989             from->nentries * png_sizeof(png_sPLT_entry));
989     if (to->entries == NULL)
990     {
-----
[ #0] Id 1, Name: "test", stopped, reason: SINGLE STEP
-----
[ #0] 0x7ffff7bb3ba8->png_set_sPLT(png_ptr=0x606420, info_ptr=0x60ca40, entries=0x7fffffdd50, nentries=0x1)
[ #1] 0x7ffff7bb7404->png_handle_sPLT(png_ptr=0x606420, info_ptr=0x60ca40, length=0x16)
[ #2] 0x7ffff7bbe62f->png_read_info(png_ptr=0x606420, info_ptr=0x60ca40)
[ #3] 0x4028ea->test_one_file(inname=0x7fffff60e "splt.png", outname=0x7fffff617 "p.png")
[ #4] 0x403c2a->main(argc=0x3, argv=0x7fffff378)
-----
985             from->nentries * png_sizeof(png_sPLT_entry));
gef> x/4xg 0x000000000060ce70
0x60ce70:      0x6161616161616161      0x00007ffff7ba3b98
0x60ce80:      0x000000000060d2f0      0x00007ffff7ba3b78
gef> p to->name
$8 = (png_charp) 0x60ce70 "aaaaaaa\230;\272\367\377\177"
gef> █
```



修复

```
png_strncpy(to->name, from->name, png_strlen(from->name)+1);
```

[CVE-2007-5269](#)

公告地址

<https://sourceforge.net/p/png-mng/mailman/png-mng-implement/thread/3.0.6.32.20071004082318.012a7628@mail.comcast.net/>

zTXt

分析

测试用例

■■\libpng\ztxt.png

漏洞代码

```
void /* PRIVATE */
png_handle_zTXt(png_structp png_ptr, png_infop info_ptr, png_uint_32 length)
{
    png_crc_read(png_ptr, (png_bytep)chunkdata, slength);
    if (png_crc_finish(png_ptr, 0))
    {
        png_free(png_ptr, chunkdata);
        return;
    }

    chunkdata[slength] = 0x00;

    for (text = chunkdata; *text; text++)
        /* empty loop */ ;

    /* zTXt must have some text after the chunkdataword */
    if (text == chunkdata + slength - 1)
    {
        png_warning(png_ptr, "Truncated zTXt chunk");
```

```

    png_free(png_ptr, chunkdata);
    return;
}

```

首先读取 chunkdata，然后末尾填 '\x00'，然后会在 chunkdata 开始位置找字符串

```

for (text = chunkdata; *text; text++)
    /* empty loop */ ;

```

后面的判断条件出现了问题

```

if (text == chunkdata + slength - 1)

```

当 chunkdata 中的字符全部都不是 '\x00' 时，text 会等于 chunkdata + slength

```

1993     for (text = chunkdata; *text; text++)
1994         /* empty loop */ ;
1995
1996     /* zTXt must have some text after the chunkdataword */
->1997     if (text == chunkdata + slength - 1)
1998     {
1999         png_warning(png_ptr, "Truncated zTXt chunk");
2000         png_free(png_ptr, chunkdata);
2001         return;
2002     }

```

```

[#0] Id 1, Name: "test", stopped, reason: SINGLE STEP

```

```

[#0] 0x7ffff7bb8469->png_handle_zTXt(png_ptr=0x606880, info_ptr=0x60cea0, length=0x8)
[#1] 0x7ffff7bbe3e5->png_read_info(png_ptr=0x606880, info_ptr=0x60cea0)
[#2] 0x4028ea->test_one_file(inname=0x7ffffffffffe60e "ztxt.png", outname=0x7ffffffffffe617 "o.png")
[#3] 0x403c2a->main(argc=0x3, argv=0x7ffffffffffe378)

```

```

1997     if (text == chunkdata + slength - 1)
gef> p text
$2 = (png_charp) 0x60e638 ""
gef> p chunkdata
$3 = (png_charp) 0x60e630 "zzzzzzzz"
gef> p text == chunkdata + slength - 1
$4 = 0x0
gef> p chunkdata + slength - 1
$5 = 0x60e637 "z"
gef>

```



后面就会越界读了。

修复

```

/* zTXt must have some text after the chunkdataword */
if (text >= chunkdata + slength - 2)
{
    png_warning(png_ptr, "Truncated zTXt chunk");
    png_free(png_ptr, chunkdata);
    return;
}

```

总结：用 == 号来判断是否出现数组越界是不安全的

sCAL

测试用例

■■\libpng\scal.png

分析

漏洞代码

```

png_handle_sCAL(png_structp png_ptr, png_infop info_ptr, png_uint_32 length)
{
    png_crc_read(png_ptr, (png_bytep)buffer, slength);
    buffer[slength] = 0x00; /* null terminate the last string */
    ep = buffer + 1;        /* skip unit byte */
    width = png_strtod(png_ptr, ep, &vp);
    if (*vp)
    {

```

```

    png_warning(png_ptr, "malformed width string in sCAL chunk");
    return;
}
for (ep = buffer; *ep; ep++)
    /* empty loop */ ;
ep++;

```

当 buffer 里面的每个字符都不是 \x00 时，最后会执行这一部分代码后，ep 会超过分配的内存块的大小，造成越界访问。

修复

在后面增加校验

```

if (buffer + slength < ep)
{
    png_warning(png_ptr, "Truncated sCAL chunk");
#ifdef PNG_FIXED_POINT_SUPPORTED) && \
    !defined(PNG_FLOATING_POINT_SUPPORTED)
    png_free(png_ptr, swidth);
#endif
    png_free(png_ptr, buffer);
    return;
}

```

[CVE-2008-1382](#)

测试用例

■■\libpng\unknown.png

分析

漏洞代码

```

void /* PRIVATE */
png_handle_unknown(png_structp png_ptr, png_infop info_ptr, png_uint_32 length)
{
    png_uint_32 skip = 0;

    png_ptr->unknown_chunk.data = (png_bytep)png_malloc(png_ptr, length);
    png_ptr->unknown_chunk.size = (png_size_t)length;
    png_crc_read(png_ptr, (png_bytep)png_ptr->unknown_chunk.data, length);
}

```

在处理 unknown 类型的 chunk 时，如果 length 为 0，png_malloc 会返回 0，然后后面的代码没有校验 png_malloc 的返回值直接使用，导致空指针引用。

修复

对 length 进行校验

```

if (length == 0)
    png_ptr->unknown_chunk.data = NULL;
else
{
    png_ptr->unknown_chunk.data = (png_bytep)png_malloc(png_ptr, length);
    png_crc_read(png_ptr, (png_bytep)png_ptr->unknown_chunk.data, length);
}

```

PS: 对 1.2.19 用测试样本跑时，会触发栈溢出，溢出在 strncpy 函数内部，很神奇。

[CVE-2008-3964](#)

测试用例

ztxt_off_by_one.png

分析

漏洞代码

```

void /* PRIVATE */
png_push_read_zTt(png_structp png_ptr, png_infop info_ptr)
{

    if (!(png_ptr->zstream.avail_out) || ret == Z_STREAM_END)
    {
        if (text == NULL)
        {
            text = (png_charp)png_malloc(png_ptr,
                (png_uint_32)(png_ptr->zbuf_size
                    - png_ptr->zstream.avail_out + key_size + 1));
            png_memcpy(text + key_size, png_ptr->zbuf,
                png_ptr->zbuf_size - png_ptr->zstream.avail_out);
            png_memcpy(text, key, key_size);
            text_size = key_size + png_ptr->zbuf_size -
                png_ptr->zstream.avail_out;
            *(text + text_size) = '\0';
        }
        else
        {
            png_charp tmp;

            tmp = text;
            text = (png_charp)png_malloc(png_ptr, text_size +
                (png_uint_32)(png_ptr->zbuf_size
                    - png_ptr->zstream.avail_out));
            png_memcpy(text, tmp, text_size);
            png_free(png_ptr, tmp);
            png_memcpy(text + text_size, png_ptr->zbuf,
                png_ptr->zbuf_size - png_ptr->zstream.avail_out);
            text_size += png_ptr->zbuf_size - png_ptr->zstream.avail_out;
            *(text + text_size) = '\0';
        }
    }
}

```

分配内存时

```

png_malloc(png_ptr, text_size +
    (png_uint_32)(png_ptr->zbuf_size
        - png_ptr->zstream.avail_out));

```

最后一步给解压后的字符串末尾赋值时

```

*(text + text_size) = '\0';

```

通过代码可以知道

```

text_size = text_size +
    (png_uint_32)(png_ptr->zbuf_size
        - png_ptr->zstream.avail_out)

```

典型的单字节数组越界即

```

buf[buf_length]

```

分配内存时，分配了 0x4006

```

->0x7ffff7bcccc1 <png_push_read_zTXt+337> call 0x7ffff7bb4650 <png_malloc@plt>
\-> 0x7ffff7bb4650 <png_malloc@plt+0> jmp QWORD PTR [rip+0x221cea] # 0x7ffff7dd6340
0x7ffff7bb4656 <png_malloc@plt+6> push 0x65
0x7ffff7bb465b <png_malloc@plt+11> jmp 0x7ffff7bb3ff0
0x7ffff7bb4660 <png_set_packswap@plt+0> jmp QWORD PTR [rip+0x221ce2] # 0x7ffff7dd6348
0x7ffff7bb4666 <png_set_packswap@plt+6> push 0x66
0x7ffff7bb466b <png_set_packswap@plt+11> jmp 0x7ffff7bb3ff0

```

```

png_malloc@plt (
    $rdi = 0x00000000000060c250->0x00000000000060c250->[loop detected],
    $rsi = 0x0000000000004006,
    $rdx = 0x000000000000000f
)

```

分配的内存 0x4006

```

1275         else
1276         {
1277             png_charp tmp;
1278
1279             tmp = text;
->1280             text = (png_charp)png_malloc(png_ptr, text_size +
1281                 (png_uint_32)(png_ptr->zbuf_size
1282                     - png_ptr->zstream.avail_out));
1283             png_memcpy(text, tmp, text_size);
1284             png_free(png_ptr, tmp);
1285             png_memcpy(text + text_size, png_ptr->zbuf,

```

先知社区

最后赋值 \x00 时, 使用 0x4006 作为索引 off-by-one

```

$rax : 0x0
$rbx : 0x4006 ←
$rcx : 0x0
$rdx : 0x2000
$rsp : 0x00007fffffffdee0 -> 0x00000000000060c250 -> 0x00000000000060c250 -> [loop detected]
$rbp : 0x0
$rsi : 0x00000000000060e750 -> 0x0000000000000000
$rdi : 0x000000000000634676 -> 0x00000001a9910000
$rip : 0x00007ffff7bccd17 -> <png_push_read_zTXt+423> mov BYTE PTR [r14+rbx*1], 0x0
$r8 : 0x337d71
$r9 : 0x1
$r10 : 0x28070c
$r11 : 0x00000000000060c380 -> 0x00000000000062563d -> 0x034ff0000fff0191
$r12 : 0x000000000000630670 -> 0x9d79000064646464 ("dddd"?
$r13 : 0x00000000000060c380 -> 0x00000000000062563d -> 0x034ff0000fff0191
$r14 : 0x000000000000630670 -> 0x9d79000064646464 ("dddd"?
$r15 : 0x00000000000060c250 -> 0x00000000000060c250 -> [loop detected]
$eflags: [carry PARITY adjust zero sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0033 $ss: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000 $gs: 0x0000

0x00007fffffffdee0|+0x0000: 0x00000000000060c250 -> 0x00000000000060c250 -> [loop detected] <-$rsp
0x00007fffffffdee8|+0x0008: 0x000000000000621630 -> 0x9c78000064646464 ("dddd"?
0x00007fffffffdef0|+0x0010: 0x000000000000610360 -> 0x0000000000000001
0x00007fffffffdef8|+0x0018: 0x000000000000000000
0x00007fffffffdf00|+0x0020: 0x00000000000016500000
0x00007fffffffdf08|+0x0028: 0x00000000000060c250 -> 0x00000000000060c250 -> [loop detected]
0x00007fffffffdf10|+0x0030: 0x000000000000610360 -> 0x0000000000000001
0x00007fffffffdf18|+0x0038: 0x00007fffffe28 -> 0x0000000000000000

0x7ffff7bccd06 <png_push_read_zTXt+406> mov eax, DWORD PTR [r15+0x150]
0x7ffff7bccd0d <png_push_read_zTXt+413> sub rbx, rax
0x7ffff7bccd10 <png_push_read_zTXt+416> add rbx, QWORD PTR [r15+0x1a8]
->0x7ffff7bccd17 <png_push_read_zTXt+423> mov BYTE PTR [r14+rbx*1], 0x0
0x7ffff7bccd1c <png_push_read_zTXt+428> cmp ebp, 0x1
0x7ffff7bccd1f <png_push_read_zTXt+431> jne 0x7ffff7bcccc50 <png_push_read_zTXt+224>
0x7ffff7bccd25 <png_push_read_zTXt+437> mov rdi, r13
0x7ffff7bccd28 <png_push_read_zTXt+440> call 0x7ffff7bb4640 <inflateReset@plt>

```

先知社区

这个漏洞的样本构造需要让 zTXt 的压缩数据的大小大于 0x2000, 因为 zstream.avail_out 初始值为 2000. zTXt 的压缩数据的大小大于 0x2000 时才能进入漏洞分支。

修复

分配的时候多分配一个字节

```

tmp = text;
text = (png_charp)png_malloc(png_ptr, text_size +
    (png_uint_32)(png_ptr->zbuf_size
    - png_ptr->zstream.avail_out + 1));

```


CVE-2008-5907

测试样本

iccp_longkeyword.png

分析

漏洞代码

```
png_size_t /* PRIVATE */
png_check_keyword(png_structp png_ptr, png_charp key, png_charpp new_key)
{
    key_len = strlen(key);
    .....
    .....
    if (key_len > 79)
    {
        png_warning(png_ptr, "keyword length must be 1 - 79 characters");
        new_key[79] = '\0'; // new_key ████████
        key_len = 79;
    }
}
```

当 key_len 大于 79 时，会使用

```
new_key[79] = '\0';
```

往地址写 0，注意到 new_key 是一个 char**p，所以上面的代码实际是往一个随机的位置写 8 字节的 0。

对应的汇编代码

```
lea     rsi, aKeywordLengthM ; "keyword length must be 1 - 79 character"...
mov     rdi, png_ptr        ; png_ptr
call    _png_warning
mov     qword ptr [new_key+278h], 0 // new_key[79] = '\0';
mov     eax, 4Fh ; 'O'
jmp     loc_12237
```

以 png_write_tEXt 为例

```
void /* PRIVATE */
png_write_tEXt(png_structp png_ptr, png_charp key, png_charp text,
    png_size_t text_len)
{
    if (key == NULL || (key_len = png_check_keyword(png_ptr, key, &new_key))==0)
    {
        png_warning(png_ptr, "Empty keyword in tEXt chunk");
        return;
    }
}
```

这里 new_key 是一个栈变量，当触发漏洞时，就会往 png_write_tEXt 函数栈帧某个位置写 8 字节 0。

调试时可以看到往栈里面写 0x0000000000000000

```

$rax : 0x0
$rbx : 0x000000000000e692 -> 0x004160639c780000
$rcx : 0x7fffffb1
$rdx : 0x00007ffff7ba5780 -> 0x0000000000000000
$rsp : 0x00007fffffdcc0 -> 0x000000000000a970 -> 0x00007ffff7bce5d0 -> 0x20676e7062696c20 (" libpng ")
$rbp : 0x62
$rsi : 0x1
$rdi : 0x1
$rip : 0x00007ffff7bbb388 -> <png_check_keyword+424> mov QWORD PTR [r13+0x278], 0x0
$r8 : 0x0
$r9 : 0x4f
$r10 : 0x28
$r11 : 0x246
$r12 : 0x000000000000a970 -> 0x00007ffff7bce5d0 -> 0x20676e7062696c20 (" libpng ")
$r13 : 0x00007fffffdcc0 -> 0x000000000000e630 -> 0x626262626262626b ("kbbbbbbb?")
$r14 : 0x62
$r15 : 0x000000000000e691 -> 0x4160639c7800006b ("k?")
$eflags: [carry parity adjust zero sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cfs: 0x0033 $fs: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000 $gs: 0x0000
-----
0x00007fffffdcc0|+0x0000: 0x000000000000a970 -> 0x00007ffff7bce5d0 -> 0x20676e7062696c20 <- $rsp
0x00007fffffdcc0|+0x0008: 0x00007fffffdcc50 -> 0x0000000000001274
0x00007fffffdcc0|+0x0010: 0x0000000000001274
0x00007fffffdcc0|+0x0018: 0x00007ffff7bbfd28 -> <png_default_write_data+40> cmp rbp, rax
0x00007fffffdcc0|+0x0020: 0x0000000000001274
0x00007fffffdcc0|+0x0028: 0x000000000000a970 -> 0x00007ffff7bce5d0 -> 0x20676e7062696c20
0x00007fffffdcc0|+0x0030: 0x0000000000000001
0x00007fffffdcc0|+0x0038: 0x3bd22b563b967f00
-----
0x7ffff7bbb379 <png_check_keyword+409> lea rsi, [rip+0x151a8] # 0x7ffff7bd0528
0x7ffff7bbb380 <png_check_keyword+416> mov rdi, r12
0x7ffff7bbb383 <png_check_keyword+419> call 0x7ffff7bb0230 <png_warning@plt>
->0x7ffff7bbb388 <png_check_keyword+424> mov QWORD PTR [r13+0x278], 0x0
0x7ffff7bbb393 <png_check_keyword+435> mov eax, 0x4f
0x7ffff7bbb398 <png_check_keyword+440> jmp 0x7ffff7bbb237 <png_check_keyword+87>

```



修复

正确使用指针

```

if (key_len > 79)
{
    png_warning(png_ptr, "keyword length must be 1 - 79 characters");
    (*new_key)[79] = '\0';
    key_len = 79;
}

```

CVE-2009-0040

分析

漏洞代码

```

png_read_png(png_structp png_ptr, png_info_ptr info_ptr,
              int transforms,
              voidp params)
{
    info_ptr->row_pointers = (png_bytepp)png_malloc(png_ptr,
        info_ptr->height * png_sizeof(png_bytep));
    for (row = 0; row < (int)info_ptr->height; row++)
    {
        info_ptr->row_pointers[row] = (png_bytep)png_malloc(png_ptr,
            png_get_rowbytes(png_ptr, info_ptr));
    }
}

```

这里会分配多个 row_pointer，当内存不足时，png_malloc 会使用 longjmp 去释放掉 row_pointers 数组内的指针，row_pointers 中后面的一些没有初始化的内存区域中的残留数据也有可能被当做指针而 free。

修复

分配内存前，初始化为 0

```

png_memset(info_ptr->row_pointers, 0, info_ptr->height
    * png_sizeof(png_bytep));
for (row = 0; row < (int)info_ptr->height; row++)
    info_ptr->row_pointers[row] = (png_bytep)png_malloc(png_ptr,
        png_get_rowbytes(png_ptr, info_ptr));
}

```

CVE-2009-5063

分析

漏洞代码

```
png_write_iCCP(png_structp png_ptr, png_charp name, int compression_type,
               png_charp profile, int profile_len)
{
    png_size_t name_len;
    png_charp new_name;
    compression_state comp;
    int embedded_profile_len = 0;

    if (profile == NULL)
        profile_len = 0;

    if (profile_len > 3)
        embedded_profile_len =
            ((* (png_bytep)profile) << 24) |
            ((* (png_bytep)profile + 1) << 16) |
            ((* (png_bytep)profile + 2) << 8) |
            ((* (png_bytep)profile + 3));

    if (profile_len < embedded_profile_len)
    {
        png_warning(png_ptr,
                    "Embedded profile length too large in iCCP chunk");
        return;
    }

    if (profile_len > embedded_profile_len)
    {
        png_warning(png_ptr,
                    "Truncating profile to actual length in iCCP chunk");
        profile_len = embedded_profile_len;
    }

    if (profile_len)
        profile_len = png_text_compress(png_ptr, profile,
                                       (png_size_t)profile_len, PNG_COMPRESSION_TYPE_BASE, &comp);
}
```

可以看到这里的 `profile_len` 和 `embedded_profile_len` 都是 `int` 类型，`embedded_profile_len` 从 png 图片的数据里面取出，当 `embedded_profile_len` 为负数时比如 (0xffffffff)，最终会进入

```
profile_len = embedded_profile_len;
```

之后会将 `profile_len` 传入

```
profile_len = png_text_compress(png_ptr, profile,
                               (png_size_t)profile_len, PNG_COMPRESSION_TYPE_BASE, &comp);
```

而 `png_text_compress` 接收的参数为 `png_size_t` 即无符号整数，所以会造成越界。

修复

修改类型为 `png_size_t`.

[CVE-2010-1205](#)

处理 PNG 的 IDAT 数据时会发生堆溢出。测试样本

xploit.png

[illegible]

Name	Value	Start	Size	Color	Comment
uint64 ngpid	89504E470D0A1A0Ah	0h	8h	Fg: Bg:	
> struct CHUNK chunk[0]	IHDR (Critical, Public, Unsafe to Copy)	8h	19h	Fg: Bg:	
> struct CHUNK chunk[1]	sRGB (Ancillary, Public, Unsafe to Copy)	21h	Dh	Fg: Bg:	
> struct CHUNK chunk[2]	pHYs (Ancillary, Public, Safe to Copy)	2Eh	15h	Fg: Bg:	
> struct CHUNK chunk[3]	tIME (Ancillary, Public, Unsafe to Copy)	43h	13h	Fg: Bg:	
✓ struct CHUNK chunk[4]	IDAT (Critical, Public, Unsafe to Copy)	56h	15F9h	Fg: Bg:	
uint32 length	15EDh	56h	4h	Fg: Bg:	
> union CTYPE type	IDAT	5Ah	4h	Fg: Bg:	
> ubyte data[5613]		5Eh	15EDh	Fg: Bg:	
uint32 crc	E4729CE1h	164Bh	4h	Fg: Bg:	
> struct CHUNK chunk[5]	IEND (Critical, Public, Unsafe to Copy)	164Fh	Ch	Fg: Bg:	

分析

```
static void rpng2_x_init(void)
{
    rpng2_info.image_data = (uch *)malloc(rowbytes * rpng2_info.height); // 0xaf0
    rpng2_info.row_pointers = (uch **)malloc(rpng2_info.height * sizeof(uch *)); // 0xaf0 0x1000 0x1000 malloc
```

以上图为例, `png2_info.height` 为 1, 首先会分配 `rowbytes` 的空间用来存储所有的 IDAT 数据, 然后会分配 1 个指针数组 `row_pointers`, 用来保存指向保存每一行数据的内存区域。其中 `rowbytes` 是通过 IHDR 里面的字段计算出来的

还是上图为例，最终计算的结果为 0xaf0。之后程序会每次读取 0xaf0 数据，然后从 `rpng2_info.row_pointers` 取出一个指针，然后往指针对应的内存空间里面写数据，直到读取完所有的 IDAT 数据。后面会使用越界的指针进行内存拷贝，导致内存写。

在溢出前, `row_pointers[1]` 后面有残留的内存指针, 因为 `row_pointers` 的分配使用的是 `malloc`, 所以会有内存残留。0x612020 是一个堆上的指针。

```
gef> p mainprog_ptr->row_pointers
$3 = (uch **) 0x612b30
gef> x/4xg 0x612b30
0x612b30: 0x0000000000061dc90 0x00000000000612020
0x612b40: 0x0000000000000020 0x0000000000000021
gef> x/4xg 0x612b20
0x612b20: 0x0000000000000000 0x0000000000000021
0x612b30: 0x0000000000061dc90 0x00000000000612020
gef> x/8xg 0x00000000000612020
0x612020: 0x0000000000000000 0x0000000000000021
0x612030: 0x00000000000611840 0x00007ffff733cb88
0x612040: 0x0000000000000020 0x00000000000002b0
0x612050: 0x00a80000000b0001 0x0020000000001770
gef>
```

内存中残留指针数据

执行完毕后会触发堆溢出把堆上的数据给覆盖了。

```
gef> x/8xg 0x00000000000612020
0x612020: 0x4141414141414100 0x4141414141414141
0x612030: 0x4141414141414141 0x4141414141414141
0x612040: 0x4141414141414141 0x4141414141414141
0x612050: 0x4141414141414141 0x4141414141414141
gef>
```

总结：分配内存空间时使用的是 png 图片中的字段，然后实际使用的空间是根据数据长度进行计算的，两者的不一致导致了漏洞。

修复

在 readpng2_row_callback 对 row_num 进行判断。

CVE-2011-2692

分析

漏洞代码

```
void /* PRIVATE */
png_handle_sCAL(png_structp png_ptr, png_infop info_ptr, png_uint_32
length) {
    png_charp ep;
    ...
    png_ptr->chunkdata = (png_charp)png_malloc_warn(png_ptr, length + 1);
    ...
    slength = (png_size_t)length;
    ...
    png_ptr->chunkdata[slength] = 0x00; /* Null terminate the last
string */

    ep = png_ptr->chunkdata + 1; /* Skip unit byte */
    ...
    width = png_strtod(png_ptr, ep, &vp);
    ...
    swidth = (png_charp)png_malloc_warn(png_ptr, png_strlen(ep) + 1);
    --
}
```

当 length 为 0 时，ep 会出现越界访问。

修复

对 length 检查

libpng.rar (0.082 MB) [下载附件](#)

点击收藏 | 0 关注 | 1

[上一篇：LLVM初探](#) [下一篇：Rubyzip库 路径遍历导致Ru...](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)