Silver Peak Security Advisory

GHOST Vulnerability. glibc: __nss_hostname_digits_dots() heap-based buffer overflow, Published by NIST on 01/28/2015

CVE-2015-0235

Summary:

US-CERT/NIST advisory for CVE-2015-0235 is dated 01/28/2015. The advisory is about a potential for buffer overflow in the gethostbyname() function in glibc.

Heap-based buffer overflow in the __nss_hostname_digits_dots function in glibc 2.2, and other 2.x versions before 2.18, allows context-dependent attackers to execute arbitrary code via vectors related to the (1) gethostbyname or (2) gethostbyname2 function, also known as "GHOST".

Silver Peak VXOA appliances are exposed to this vulnerability, and the patch to resolve this is detailed under the heading, Resolution.

Details:

CVE provides information on the advisory and is located at:
https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-0235

The full advisory located at http://www.openwall.com/lists/oss-security/2015/01/27/9 reads as follows:

```
--[ 1 - Summary ]-----------------------------------------------------------
-

During a code audit performed internally at Qualys, we discovered a
buffer overflow in the __nss_hostname_digits_dots() function of the GNU
C Library (glibc). This bug is reachable both locally and remotely via
the gethostbyname*() functions, so we decided to analyze it -- and its
impact -- thoroughly, and named this vulnerability "GHOST".

Our main conclusions are:

- Via gethostbyname() or gethostbyname2(), the overflowed buffer is
  located in the heap. Via gethostbyname_r() or gethostbyname2_r(), the
```

overflowed buffer is caller-supplied (and may therefore be located in the heap, stack, .data, .bss, etc; however, we have seen no such call in practice).

- At most sizeof(char *) bytes can be overwritten (ie, 4 bytes on 32-bit machines, and 8 bytes on 64-bit machines). Bytes can be overwritten only with digits ('0'...'9'), dots ('.'), and a terminating null character ('\0').

- Despite these limitations, arbitrary code execution can be achieved. As a proof of concept, we developed a full-fledged remote exploit against the Exim mail server, bypassing all existing protections (ASLR, PIE, and NX) on both 32-bit and 64-bit machines. We will publish our exploit as a Metasploit module in the near future.

- The first vulnerable version of the GNU C Library is glibc-2.2, released on November 10, 2000.

- We identified a number of factors that mitigate the impact of this bug. In particular, we discovered that it was fixed on May 21, 2013 (between the releases of glibc-2.17 and glibc-2.18). Unfortunately, it was not recognized as a security threat; as a result, most stable and long-term-support distributions were left exposed (and still are): Debian 7 (wheezy), Red Hat Enterprise Linux 6 & 7, CentOS 6 & 7, Ubuntu 12.04, for example.

--[2 - Analysis]---

The vulnerable function, __nss_hostname_digits_dots(), is called internally by the glibc in nss/getXXbyYY.c (the non-reentrant version) and nss/getXXbyYY_r.c (the reentrant version). However, the calls are surrounded by #ifdef HANDLE_DIGITS_DOTS, a macro defined only in:

- inet/gethstbynm.c
- inet/gethstbynm2.c
- inet/gethstbynm_r.c
- inet/gethstbynm2_r.c
- nscd/gethstbynm3_r.c

These files implement the gethostbyname*() family, and hence the only way to reach __nss_hostname_digits_dots() and its buffer overflow. The purpose of this function is to avoid expensive DNS lookups if the hostname argument is already an IPv4 or IPv6 address.

The code below comes from glibc-2.17:

```
 35 int
 36 __nss_hostname_digits_dots (const char *name, struct hostent *resbuf,
 37                              char **buffer, size_t *buffer_size,
 38                              size_t buflen, struct hostent **result,
 39                              enum nss_status *status, int af, int
*h_errnop)
 40 {
 ..
 57   if (isdigit (name[0]) || isxdigit (name[0]) || name[0] == ':')
```

```
 58     {
 59       const char *cp;
 60       char *hostname;
 61       typedef unsigned char host_addr_t[16];
 62       host_addr_t *host_addr;
 63       typedef char *host_addr_list_t[2];
 64       host_addr_list_t *h_addr_ptrs;
 65       char **h_alias_ptr;
 66       size_t size_needed;
..
 85       size_needed = (sizeof (*host_addr)
 86                      + sizeof (*h_addr_ptrs) + strlen (name) + 1);
 87
 88       if (buffer_size == NULL)
 89         {
 90           if (buflen < size_needed)
 91             {
..
 95               goto done;
 96             }
 97         }
 98       else if (buffer_size != NULL && *buffer_size < size_needed)
 99         {
100           char *new_buf;
101           *buffer_size = size_needed;
102           new_buf = (char *) realloc (*buffer, *buffer_size);
103
104           if (new_buf == NULL)
105             {
...
114               goto done;
115             }
116           *buffer = new_buf;
117         }
...
121       host_addr = (host_addr_t *) *buffer;
122       h_addr_ptrs = (host_addr_list_t *)
123         ((char *) host_addr + sizeof (*host_addr));
124       h_alias_ptr = (char **) ((char *) h_addr_ptrs + sizeof
(*h_addr_ptrs));
125       hostname = (char *) h_alias_ptr + sizeof (*h_alias_ptr);
126
127       if (isdigit (name[0]))
128         {
129           for (cp = name;; ++cp)
130             {
131               if (*cp == '\0')
132                 {
133                   int ok;
134
135                   if (*--cp == '.')
136                     break;
...
142                   if (af == AF_INET)
143                     ok = __inet_aton (name, (struct in_addr *)
host_addr);
144                   else
```

```
145                     {
146                       assert (af == AF_INET6);
147                       ok = inet_pton (af, name, host_addr) > 0;
148                     }
149                   if (! ok)
150                     {
...
154                       goto done;
155                     }
156
157                   resbuf->h_name = strcpy (hostname, name);
...
194                   goto done;
195                 }
196
197               if (!isdigit (*cp) && *cp != '.')
198                 break;
199             }
200         }
...
```

Lines 85-86 compute the size_needed to store three (3) distinct entities in buffer: host_addr, h_addr_ptrs, and name (the hostname). Lines 88-117 make sure the buffer is large enough: lines 88-97 correspond to the reentrant case, lines 98-117 to the non-reentrant case.

Lines 121-125 prepare pointers to store four (4) distinct entities in buffer: host_addr, h_addr_ptrs, h_alias_ptr, and hostname. The sizeof (*h_alias_ptr) -- the size of a char pointer -- is missing from the computation of size_needed.

The strcpy() on line 157 should therefore allow us to write past the end of buffer, at most (depending on strlen(name) and alignment) 4 bytes on 32-bit machines, or 8 bytes on 64-bit machines. There is a similar strcpy() after line 200, but no buffer overflow:

```
236           size_needed = (sizeof (*host_addr)
237                          + sizeof (*h_addr_ptrs) + strlen (name) + 1);
...
267           host_addr = (host_addr_t *) *buffer;
268           h_addr_ptrs = (host_addr_list_t *)
269             ((char *) host_addr + sizeof (*host_addr));
270           hostname = (char *) h_addr_ptrs + sizeof (*h_addr_ptrs);
...
289                   resbuf->h_name = strcpy (hostname, name);
```

In order to reach the overflow at line 157, the hostname argument must meet the following requirements:

- Its first character must be a digit (line 127).

- Its last character must not be a dot (line 135).

- It must comprise only digits and dots (line 197) (we call this the "digits-and-dots" requirement).

- It must be long enough to overflow the buffer. For example, the

```
  non-reentrant gethostbyname*() functions initially allocate their
  buffer with a call to malloc(1024) (the "1-KB" requirement).

- It must be successfully parsed as an IPv4 address by inet_aton() (line
  143), or as an IPv6 address by inet_pton() (line 147). Upon careful
  analysis of these two functions, we can further refine this
  "inet-aton" requirement:

  . It is impossible to successfully parse a "digits-and-dots" hostname
    as an IPv6 address with inet_pton() (':' is forbidden). Hence it is
    impossible to reach the overflow with calls to gethostbyname2() or
    gethostbyname2_r() if the address family argument is AF_INET6.

  . Conclusion: inet_aton() is the only option, and the hostname must
    have one of the following forms: "a.b.c.d", "a.b.c", "a.b", or "a",
    where a, b, c, d must be unsigned integers, at most 0xffffffful,
    converted successfully (ie, no integer overflow) by strtoul() in
    decimal or octal (but not hexadecimal, because 'x' and 'X' are
    forbidden).
```

--snip--

NIST has added the vulnerability summary for this CVE to their National Cyber Awareness System database:

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-0235

Recommended Action for Silver Peak Customers:

Silver Peak VXOA appliances:
Silver Peak VXOA appliances are affected by this vulnerability. To mitigate risk, Silver Peak recommends upgrading VXOA appliances to the releases listed in RESOLUTION. The patch is in line with the recommendation in the CVE-2015-0235 advisory.

Resolution:

Silver Peak Issue Id 25279 tracks this vulnerability.

The resolution for this vulnerability is in each of the following release branches:

- **VXOA 6.0.11.0 and later releases**
- **VXOA 6.2.8.0 and later releases**
- **VXOA 7.1.1.0 and later releases**
- **VXOA 7.2.0.0 and later releases**