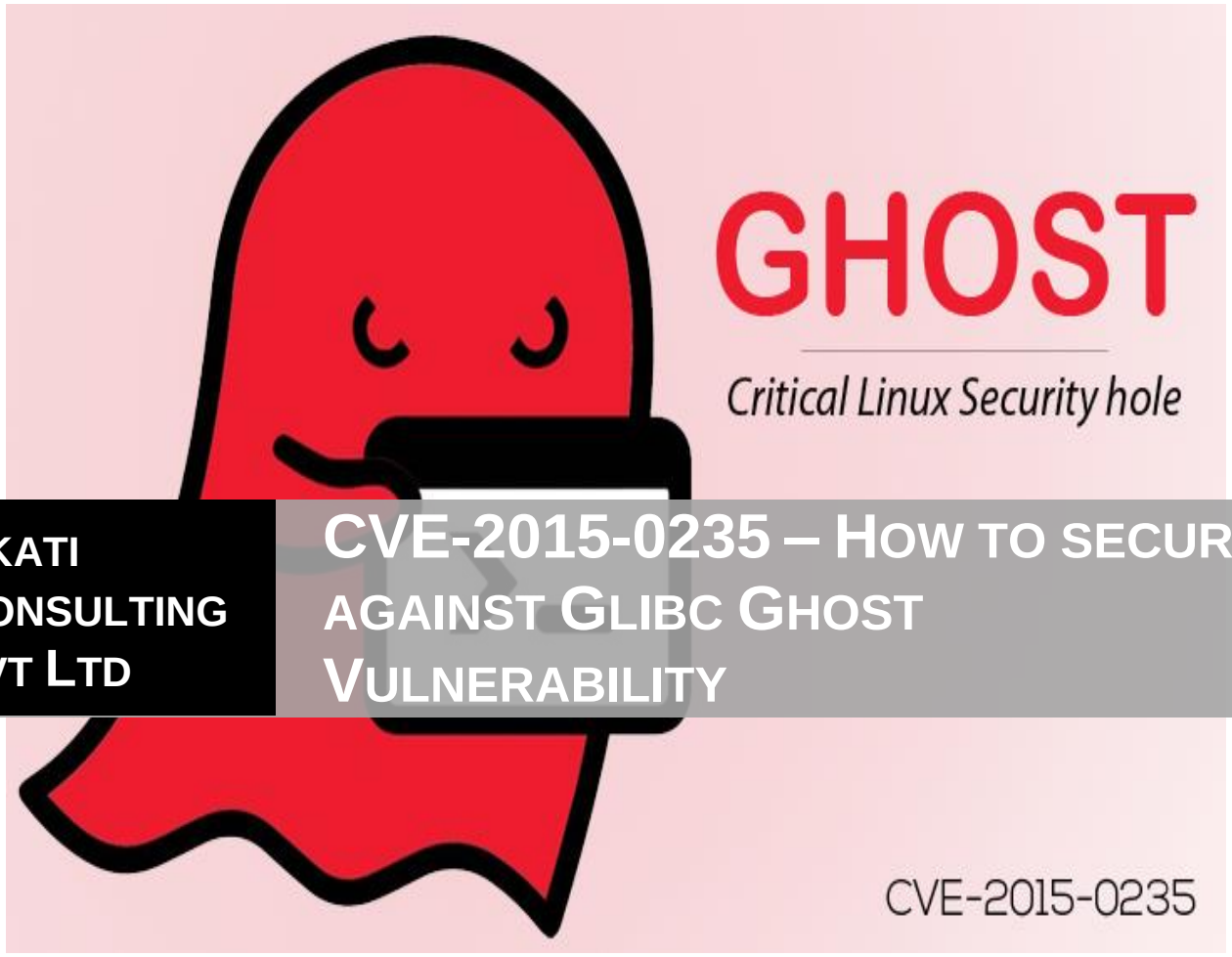


AKATI CONSULTING PVT LTD

# CVE-2015-0235 – HOW TO SECURE AGAINST GLIBC GHOST VULNERABILITY

**Author | Rajivarnan.R Cyber security Researcher at Akati consulting Pvt Ltd**

# Description

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, aka "GHOST."

The GHOST vulnerability is a serious weakness in the Linux glibc library. It allows attackers to remotely take complete control of the victim system without having any prior knowledge of system credentials. CVE-2015-0235 has been assigned to this issue.

Qualys security researchers discovered this bug and worked closely with Linux distribution vendors. And as a result of that we are releasing this advisory today as a coordinated effort, and patches for all distribution are available January 27, 2015.

# What is glibc?

The GNU C Library or glibc is an implementation of the standard C library and a core part of the Linux operating system. Without this library a Linux system will not function.

# What is the vulnerability?

During a code audit Qualys researchers discovered a buffer overflow in the __nss_hostname_digits_dots() function of glibc. This bug can be triggered both locally and remotely via all the gethostbyname*() functions. Applications have access to the DNS resolver primarily through the gethostbyname*() set of functions. These functions convert a hostname into an IP address.

During a code audit performed internally at Qualys, 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".

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 size of(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.

# 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() (line143), 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() orgethostbyname2_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).

# Impact of this Bug

The impact of this bug is reduced significantly by the following
Reasons:

- A patch already exists (since May 21, 2013), and has been applied and tested since glibc-2.18, released on August 12, 2013:

[BZ #15014]
- nss/getXXbyYY_r.c (INTERNAL (REENTRANT_NAME))

[HANDLE_DIGITS_DOTS]: Set any_service when digits-dots parsing was successful.
- nss/digits_dots.c (__nss_hostname_digits_dots): Remove redundant variable declarations and reallocation of buffer when parsing as IPv6 address. Always set NSS status when called from reentrant functions. Use NETDB_INTERNAL instead of TRY_AGAIN when buffer too small. Correct computation of needed size.
- nss/Makefile (tests): Add test-digits-dots.
- nss/test-digits-dots.c: New test.

- The gethostbyname*() functions are obsolete; with the advent of IPv6, recent applications use getaddrinfo() instead.

- Many programs, especially SUID binaries reachable locally, use gethostbyname() if, and only if, a preliminary call to inet_aton() fails. However, a subsequent call must also succeed (the "inet-aton" requirement) in order to reach the overflow: this is impossible, and such programs are therefore safe.

- Most of the other programs, especially servers reachable remotely, use gethostbyname() to perform forward-confirmed reverse DNS (FCrDNS, also known as full-circle reverse DNS) checks. These programs are generally safe, because the hostname passed to gethostbyname() has normally been pre-validated by DNS software:

"a string of labels each containing up to 63 8-bit octets, separated by dots, and with a maximum total of 255 octets." This makes it impossible to satisfy the "1-KB" requirement.

Actually, glibc's DNS resolver can produce hostnames of up to(almost) 1025 characters (in case of bit-string labels, and special or non-printable characters). But this introduces backslashes ('\\') and makes it impossible to satisfy the "digits-and-dots" requirement.

# Case Study

In this section, we will analyze real-world examples of programs that call the gethostbyname*() functions, but we first introduce a small test program that checks whether a system is vulnerable or not:

```
[user () fedora-19 ~]$ cat > GHOST.c << EOF
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#define CANARY "in_the_coal_mine"

struct {
  char buffer[1024];
  char canary[sizeof(CANARY)];
} temp = { "buffer", CANARY };

int main(void) {
  struct hostent resbuf;
  struct hostent *result;
  int herrno;
  int retval;

  /*** strlen (name) = size_needed - sizeof (*host_addr) - sizeof (*h_addr_ptrs) - 1; ***/
  size_t len = sizeof(temp.buffer) - 16*sizeof(unsigned char) - 2*sizeof(char *) - 1;
  char name[sizeof(temp.buffer)];
  memset(name, '0', len);
  name[len] = '\0';

  retval = gethostbyname_r(name, &resbuf, temp.buffer, sizeof(temp.buffer), &result, &herrno);

  if (strcmp(temp.canary, CANARY) != 0) {
    puts("vulnerable");
    exit(EXIT_SUCCESS);
  }
  if (retval == ERANGE) {
    puts("not vulnerable");
    exit(EXIT_SUCCESS);
  }
  puts("should not happen");
  exit(EXIT_FAILURE);
}
EOF
```

[user () fedora-19 ~]$ gcc GHOST.c -o GHOST

On Fedora 19 (glibc-2.17):

[user () fedora-19 ~]$ ./GHOST
vulnerable

On Fedora 20 (glibc-2.18):

[user () fedora-20 ~]$ ./GHOST
not vulnerable

## What versions and operating systems are affected?

The first vulnerable version of the GNU C Library affected by this 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 including Debian 7 (wheezy), Red Hat Enterprise Linux 6 & 7, CentOS 6 & 7, Ubuntu 12.04, for example.

## Following are the potentially exploitable services

- Procmail
- Exim
- Pppd
- Clockdiff

You can find the list of services which are rely on the GNU C libraries by executing following command

**Code:**
**lsof | grep libc | awk '{print $1}' | sort | uni**

## Fix for Centos/RHEL/Fedora 5,6,7

**Code:**

```
yum update glibc
sudo restart
```

## Fix for Ubuntu

**Code:**

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo restart
```

**Thank you**