



**AKATI
CONSULTING
PVT LTD**

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

CVE-2015-0235



**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/getstbynm.c`
- `inet/getstbynm2.c`
- `inet/getstbynm_r.c`
- `inet/getstbynm2_r.c`
- `nscd/getstbynm3_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 {
41     ..
42     ..
43     ..
44     ..
45     ..
46     ..
47     ..
48     ..
49     ..
50     ..
51     ..
52     ..
53     ..
54     ..
55     ..
56     ..
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;
67     }
68     ..
69     ..
70     ..
71     ..
72     ..
73     ..
74     ..
75     ..
76     ..
77     ..
78     ..
79     ..
80     ..
81     ..
82     ..
83     ..
84     ..
85     ..
86     ..
87     ..
88     ..
89     ..
90     ..
91     ..
92     ..
93     ..
94     ..
95     ..
96     ..
97     ..
98     ..
99     ..
100    ..
101    ..
102    ..
103    ..
104    ..
105    ..
106    ..
107    ..
108    ..
109    ..
110    ..
111    ..
112    ..
113    ..
114    ..
115    ..
116    ..
117    ..
118    ..
119    ..
120    ..
121    ..
122    ..
123    ..
124    ..
125    ..
126    ..
127    ..
128    ..
129    ..
130    ..
131    ..
132    ..
133    ..
134    ..
135    ..
136    ..
137    ..
138    ..
139    ..
140    ..
141    ..
142    ..
143    ..
144    ..
145    ..
146    ..
147    ..
148    ..
149    ..
150    ..
151    ..
152    ..
153    ..
154    ..
155    ..
156    ..
157    ..
158    ..
159    ..
160    ..
161    ..
162    ..
163    ..
164    ..
165    ..
166    ..
167    ..
168    ..
169    ..
170    ..
171    ..
172    ..
173    ..
174    ..
175    ..
176    ..
177    ..
178    ..
179    ..
180    ..
181    ..
182    ..
183    ..
184    ..
185    ..
186    ..
187    ..
188    ..
189    ..
190    ..
191    ..
192    ..
193    ..
194    ..
195    ..
196    ..
197    ..
198    ..
199    ..
200    ..
201    ..
202    ..
203    ..
204    ..
205    ..
206    ..
207    ..
208    ..
209    ..
210    ..
211    ..
212    ..
213    ..
214    ..
215    ..
216    ..
217    ..
218    ..
219    ..
220    ..
221    ..
222    ..
223    ..
224    ..
225    ..
226    ..
227    ..
228    ..
229    ..
230    ..
231    ..
232    ..
233    ..
234    ..
235    ..
236    ..
237    ..
238    ..
239    ..
240    ..
241    ..
242    ..
243    ..
244    ..
245    ..
246    ..
247    ..
248    ..
249    ..
250    ..
251    ..
252    ..
253    ..
254    ..
255    ..
256    ..
257    ..
258    ..
259    ..
260    ..
261    ..
262    ..
263    ..
264    ..
265    ..
266    ..
267    ..
268    ..
269    ..
270    ..
271    ..
272    ..
273    ..
274    ..
275    ..
276    ..
277    ..
278    ..
279    ..
280    ..
281    ..
282    ..
283    ..
284    ..
285    ..
286    ..
287    ..
288    ..
289    ..
290    ..
291    ..
292    ..
293    ..
294    ..
295    ..
296    ..
297    ..
298    ..
299    ..
300    ..
301    ..
302    ..
303    ..
304    ..
305    ..
306    ..
307    ..
308    ..
309    ..
310    ..
311    ..
312    ..
313    ..
314    ..
315    ..
316    ..
317    ..
318    ..
319    ..
320    ..
321    ..
322    ..
323    ..
324    ..
325    ..
326    ..
327    ..
328    ..
329    ..
330    ..
331    ..
332    ..
333    ..
334    ..
335    ..
336    ..
337    ..
338    ..
339    ..
340    ..
341    ..
342    ..
343    ..
344    ..
345    ..
346    ..
347    ..
348    ..
349    ..
350    ..
351    ..
352    ..
353    ..
354    ..
355    ..
356    ..
357    ..
358    ..
359    ..
360    ..
361    ..
362    ..
363    ..
364    ..
365    ..
366    ..
367    ..
368    ..
369    ..
370    ..
371    ..
372    ..
373    ..
374    ..
375    ..
376    ..
377    ..
378    ..
379    ..
380    ..
381    ..
382    ..
383    ..
384    ..
385    ..
386    ..
387    ..
388    ..
389    ..
390    ..
391    ..
392    ..
393    ..
394    ..
395    ..
396    ..
397    ..
398    ..
399    ..
400    ..
401    ..
402    ..
403    ..
404    ..
405    ..
406    ..
407    ..
408    ..
409    ..
410    ..
411    ..
412    ..
413    ..
414    ..
415    ..
416    ..
417    ..
418    ..
419    ..
420    ..
421    ..
422    ..
423    ..
424    ..
425    ..
426    ..
427    ..
428    ..
429    ..
430    ..
431    ..
432    ..
433    ..
434    ..
435    ..
436    ..
437    ..
438    ..
439    ..
440    ..
441    ..
442    ..
443    ..
444    ..
445    ..
446    ..
447    ..
448    ..
449    ..
450    ..
451    ..
452    ..
453    ..
454    ..
455    ..
456    ..
457    ..
458    ..
459    ..
460    ..
461    ..
462    ..
463    ..
464    ..
465    ..
466    ..
467    ..
468    ..
469    ..
470    ..
471    ..
472    ..
473    ..
474    ..
475    ..
476    ..
477    ..
478    ..
479    ..
480    ..
481    ..
482    ..
483    ..
484    ..
485    ..
486    ..
487    ..
488    ..
489    ..
490    ..
491    ..
492    ..
493    ..
494    ..
495    ..
496    ..
497    ..
498    ..
499    ..
500    ..
501    ..
502    ..
503    ..
504    ..
505    ..
506    ..
507    ..
508    ..
509    ..
510    ..
511    ..
512    ..
513    ..
514    ..
515    ..
516    ..
517    ..
518    ..
519    ..
520    ..
521    ..
522    ..
523    ..
524    ..
525    ..
526    ..
527    ..
528    ..
529    ..
530    ..
531    ..
532    ..
533    ..
534    ..
535    ..
536    ..
537    ..
538    ..
539    ..
540    ..
541    ..
542    ..
543    ..
544    ..
545    ..
546    ..
547    ..
548    ..
549    ..
550    ..
551    ..
552    ..
553    ..
554    ..
555    ..
556    ..
557    ..
558    ..
559    ..
560    ..
561    ..
562    ..
563    ..
564    ..
565    ..
566    ..
567    ..
568    ..
569    ..
570    ..
571    ..
572    ..
573    ..
574    ..
575    ..
576    ..
577    ..
578    ..
579    ..
580    ..
581    ..
582    ..
583    ..
584    ..
585    ..
586    ..
587    ..
588    ..
589    ..
590    ..
591    ..
592    ..
593    ..
594    ..
595    ..
596    ..
597    ..
598    ..
599    ..
600    ..
601    ..
602    ..
603    ..
604    ..
605    ..
606    ..
607    ..
608    ..
609    ..
610    ..
611    ..
612    ..
613    ..
614    ..
615    ..
616    ..
617    ..
618    ..
619    ..
620    ..
621    ..
622    ..
623    ..
624    ..
625    ..
626    ..
627    ..
628    ..
629    ..
630    ..
631    ..
632    ..
633    ..
634    ..
635    ..
636    ..
637    ..
638    ..
639    ..
640    ..
641    ..
642    ..
643    ..
644    ..
645    ..
646    ..
647    ..
648    ..
649    ..
650    ..
651    ..
652    ..
653    ..
654    ..
655    ..
656    ..
657    ..
658    ..
659    ..
660    ..
661    ..
662    ..
663    ..
664    ..
665    ..
666    ..
667    ..
668    ..
669    ..
670    ..
671    ..
672    ..
673    ..
674    ..
675    ..
676    ..
677    ..
678    ..
679    ..
680    ..
681    ..
682    ..
683    ..
684    ..
685    ..
686    ..
687    ..
688    ..
689    ..
690    ..
691    ..
692    ..
693    ..
694    ..
695    ..
696    ..
697    ..
698    ..
699    ..
700    ..
701    ..
702    ..
703    ..
704    ..
705    ..
706    ..
707    ..
708    ..
709    ..
710    ..
711    ..
712    ..
713    ..
714    ..
715    ..
716    ..
717    ..
718    ..
719    ..
720    ..
721    ..
722    ..
723    ..
724    ..
725    ..
726    ..
727    ..
728    ..
729    ..
730    ..
731    ..
732    ..
733    ..
734    ..
735    ..
736    ..
737    ..
738    ..
739    ..
740    ..
741    ..
742    ..
743    ..
744    ..
745    ..
746    ..
747    ..
748    ..
749    ..
750    ..
751    ..
752    ..
753    ..
754    ..
755    ..
756    ..
757    ..
758    ..
759    ..
760    ..
761    ..
762    ..
763    ..
764    ..
765    ..
766    ..
767    ..
768    ..
769    ..
770    ..
771    ..
772    ..
773    ..
774    ..
775    ..
776    ..
777    ..
778    ..
779    ..
780    ..
781    ..
782    ..
783    ..
784    ..
785    ..
786    ..
787    ..
788    ..
789    ..
790    ..
791    ..
792    ..
793    ..
794    ..
795    ..
796    ..
797    ..
798    ..
799    ..
800    ..
801    ..
802    ..
803    ..
804    ..
805    ..
806    ..
807    ..
808    ..
809    ..
810    ..
811    ..
812    ..
813    ..
814    ..
815    ..
816    ..
817    ..
818    ..
819    ..
820    ..
821    ..
822    ..
823    ..
824    ..
825    ..
826    ..
827    ..
828    ..
829    ..
830    ..
831    ..
832    ..
833    ..
834    ..
835    ..
836    ..
837    ..
838    ..
839    ..
840    ..
841    ..
842    ..
843    ..
844    ..
845    ..
846    ..
847    ..
848    ..
849    ..
850    ..
851    ..
852    ..
853    ..
854    ..
855    ..
856    ..
857    ..
858    ..
859    ..
860    ..
861    ..
862    ..
863    ..
864    ..
865    ..
866    ..
867    ..
868    ..
869    ..
870    ..
871    ..
872    ..
873    ..
874    ..
875    ..
876    ..
877    ..
878    ..
879    ..
880    ..
881    ..
882    ..
883    ..
884    ..
885    ..
886    ..
887    ..
888    ..
889    ..
890    ..
891    ..
892    ..
893    ..
894    ..
895    ..
896    ..
897    ..
898    ..
899    ..
900    ..
901    ..
902    ..
903    ..
904    ..
905    ..
906    ..
907    ..
908    ..
909    ..
910    ..
911    ..
912    ..
913    ..
914    ..
915    ..
916    ..
917    ..
918    ..
919    ..
920    ..
921    ..
922    ..
923    ..
924    ..
925    ..
926    ..
927    ..
928    ..
929    ..
930    ..
931    ..
932    ..
933    ..
934    ..
935    ..
936    ..
937    ..
938    ..
939    ..
940    ..
941    ..
942    ..
943    ..
944    ..
945    ..
946    ..
947    ..
948    ..
949    ..
950    ..
951    ..
952    ..
953    ..
954    ..
955    ..
956    ..
957    ..
958    ..
959    ..
960    ..
961    ..
962    ..
963    ..
964    ..
965    ..
966    ..
967    ..
968    ..
969    ..
970    ..
971    ..
972    ..
973    ..
974    ..
975    ..
976    ..
977    ..
978    ..
979    ..
980    ..
981    ..
982    ..
983    ..
984    ..
985    ..
986    ..
987    ..
988    ..
989    ..
990    ..
991    ..
992    ..
993    ..
994    ..
995    ..
996    ..
997    ..
998    ..
999    ..
1000   ..
```

```
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         {
92             ..
93             goto done;
94         }
95     }
96     else if (buffer_size != NULL && *buffer_size < size_needed)
97     {
98         char *new_buf;
99         *buffer_size = size_needed;
100        new_buf = (char *) realloc (*buffer, *buffer_size);
101
102        if (new_buf == NULL)
103        {
104            ..
105            goto done;
106        }
107        *buffer = new_buf;
108    }
109
110    ..
111    host_addr = (host_addr_t *) *buffer;
112    h_addr_ptrs = (host_addr_list_t *)
113        ((char *) host_addr + sizeof (*host_addr));
114    h_alias_ptr = (char **) ((char *) h_addr_ptrs + sizeof (*h_addr_ptrs));
115    hostname = (char *) h_alias_ptr + sizeof (*h_alias_ptr);
116
117    if (isdigit (name[0]))
118    {
119        for (cp = name;; ++cp)
120        {
121            if (*cp == '\0')
122            {
123                int ok;
124
125                if (*--cp == '.')
126                    break;
127            }
128
129            if (af == AF_INET)
130                ok = __inet_aton (name, (struct in_addr *) host_addr);
131            else
132            {
133                assert (af == AF_INET6);
134                ok = inet_pton (af, name, host_addr) > 0;
135            }
136            if (! ok)
137            {
```

```

...
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 0xffffffff, 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:
lsuf | 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