Date: Tue, 19 May 2020 10:05:06 -0700
From: Qualys Security Advisory <qsa@...lys.com>
To: oss-security@...ts.openwall.com
Subject: Remote Code Execution in qmail (CVE-2005-1513)

Qualys Security Advisory

15 years later: Remote Code Execution in qmail (CVE-2005-1513)

==
Contents
==

Summary
Analysis
Exploitation
qmail-verify
- CVE-2020-3811
- CVE-2020-3812
Mitigations
Acknowledgments
Patches

==
Summary
==

TLDR: In 2005, three vulnerabilities were discovered in qmail but were never fixed because they were believed to be unexploitable in a default installation. We recently re-discovered these vulnerabilities and were able to exploit one of them remotely in a default installation.

In May 2005, Georgi Guninski published "64 bit qmail fun", three vulnerabilities in qmail (CVE-2005-1513, CVE-2005-1514, CVE-2005-1515):

    http://www.guninski.com/where_do_you_want_billg_to_go_today_4.html

Surprisingly, we re-discovered these vulnerabilities during a recent qmail audit; they have never been fixed because, as stated by qmail's author Daniel J. Bernstein (in https://cr.yp.to/qmail/guarantee.html):

    "This claim is denied. Nobody gives gigabytes of memory to each qmail-smtpd process, so there is no problem with qmail's assumption that allocated array lengths fit comfortably into 32 bits."

Indeed, the memory consumption of each qmail-smtpd process is severely limited by default (by qmail-smtpd's startup script); for example, on Debian 10 (the latest stable release), it is limited to roughly 7MB.

Unfortunately, we discovered that these vulnerabilities also affect qmail-local, which is reachable remotely and is not memory-limited by default (we investigated many qmail packages, and *all* of them limit qmail-smtpd's memory, but *none* of them limits qmail-local's memory).

As a proof of concept, we developed a reliable, local and remote exploit against Debian's qmail package in its default configuration. This proof of concept requires 4GB of disk space and 8GB of memory, and allows an attacker to execute arbitrary shell commands as any user, except root (and a few system users who do not own their home directory). We will publish our proof-of-concept exploit in the near future.

About our new discovery, Daniel J. Bernstein issues the following statement:

    "https://cr.yp.to/qmail/guarantee.html has for many years mentioned qmail's assumption that allocated array lengths fit comfortably into 32 bits. I run each qmail service under softlimit -m12345678, and I recommend the same for other installations."

Finally, we also discovered two minor vulnerabilities in qmail-verify (a third-party qmail patch that is included in, for example, Debian's qmail package): CVE-2020-3811 (a mail-address verification bypass), and CVE-2020-3812 (a local information disclosure).

==
Analysis
==

We decided to exploit Georgi Guninski's vulnerability "1. integer overflow in stralloc_readyplus" (CVE-2005-1513). There are, in fact, four potential integer overflows in stralloc_readyplus; three in the GEN_ALLOC_readyplus() macro (which generates the stralloc_readyplus() function), at line 21 (n += x->len), line 23 (x->a = base + n + ...), and line 24 (x->a * sizeof(type)):

```
------------------------------------------------------------------------
 17 #define GEN_ALLOC_readyplus(ta,type,field,len,a,i,n,x,base,ta_rplus) \
 18 int ta_rplus(x,n) register ta *x; register unsigned int n; \
 19 { register unsigned int i; \
 20   if (x->field) { \
 21     i = x->a; n += x->len; \
 22     if (n > i) { \
 23       x->a = base + n + (n >> 3); \
 24       if (alloc_re(&x->field,i * sizeof(type),x->a * sizeof(type))) return 1; \
 25       x->a = i; return 0; } \
 26     return 1; } \
 27   x->len = 0; \
 28   return !!(x->field = (type *) alloc((x->a = n) * sizeof(type))); }
------------------------------------------------------------------------
```

and, in theory, one integer overflow in the alloc() function itself (which is called by the alloc_re() function), at line 18:

```
------------------------------------------------------------------------
 14 /*@...l@...*@out@...har *alloc(n)
 15 unsigned int n;
 16 {
 17   char *x;
 18   n = ALIGNMENT + n - (n & (ALIGNMENT - 1)); /* XXX: could overflow */
 ..
 20   x = malloc(n);
 ..
 22   return x;
 23 }
------------------------------------------------------------------------
```

In practice, the integer overflows at line 21 (in GEN_ALLOC_readyplus()) and line 18 (in alloc()) are very hard to trigger; and the one at line 24 (in GEN_ALLOC_readyplus()) is irrelevant to stralloc_readyplus's case (because type is char and sizeof(type) is therefore 1).

On the other hand, the integer overflow at line 23 (in GEN_ALLOC_readyplus()) is easy to trigger, because the size x->a of the buffer is increased by one eighth every time it is re-allocated: we send a very large mail message that contains a very long header line (nearly 4GB), and this line triggers stralloc_readyplus's integer overflow while in the getln() function, which is called by the bouncexf() function, at the beginning of the qmail-local program. qmail-local is responsible for the local delivery of mail messages, and runs with the privileges of the local recipient (or qmail's "alias" user, if the local recipient is "root", for example).

After the size of the buffer is overflowed (at line 23), the alloc_re() function is called (at line 24), but with n < m, where n is the size of the new buffer y, and m is the size of the old buffer x:

```
  4 int alloc_re(x,m,n)
  5 char **x;
  6 unsigned int m;
  7 unsigned int n;
  8 {
  9   char *y;
 10
 11   y = alloc(n);
 12   if (!y) return 0;
 13   byte_copy(y,m,*x);
 14   alloc_free(*x);
 15   *x = y;
 16   return 1;
 17 }
```

In other words, we transformed stralloc_readyplus's integer overflow into an mmap-based buffer overflow at line 13 (byte_copy() is qmail's version of memcpy()): m is nearly 4GB (the length of our very long header line), but n is roughly 512MB (one eighth of m).

Exploitation

To survive this large buffer overflow, we carefully choose the number and lengths of the very first lines in our mail message (they crucially influence the sequence of buffer re-allocations that eventually lead to the integer and buffer overflows), and obtain the following mmap layout:

```
-------|-------|-----------------------------------------------|------
XXXXXXX|   y   |                       x                       | libc
-------|-------|-----------------------------------------------|------
       | 512MB |                      4GB                      |
```

Consequently, we safely overflow the new buffer y, and overwrite the malloc header of the old buffer x, with the contents of our very long header line. To exploit this malloc-header corruption when free(x) is called (at line 14), we devised an unusual method that bypasses NX and ASLR, but does not work against a full-RELRO binary (but the qmail-local binary on Debian 10 is partial-RELRO only). This does not mean, however, that a full-RELRO binary is not exploitable: other methods may exist, the only limit to malloc exploitation is the imagination.

First, we overwrite the prev_size and size fields of x's malloc header, we set its IS_MMAPPED bit to 1, and therefore enter the munmap_chunk() function in __libc_free() (where p is a pointer to x's malloc header):

```
2810 static void
2811 munmap_chunk (mchunkptr p)
2812 {
2813   INTERNAL_SIZE_T size = chunksize (p);
....
2822   uintptr_t block = (uintptr_t) p - prev_size (p);
2823   size_t total_size = prev_size (p) + size;
....
2838   __munmap ((char *) block, total_size);
2839 }
```

Because we completely control the size field (at line 2813) and the prev_size field (at lines 2822 and 2823), we completely control the block address (relative to p, and hence x) and the total_size of the __munmap() call (at line 2838). In other words, we can munmap() an arbitrary mmap region, without knowing the ASLR; we munmap() roughly 576MB at the end of x, including the first few pages of the libc:

```
-------|-------|-----------------------------------------|-------+-|----
XXXXXXX|   y   |                       x                 |XXXXXXXXX|ibc
-------|-------|-----------------------------------------|-------+-|----
```

The first pages of the libc do not actually contain executable code: they contain the ELF .dynsym section, which associates a symbol (for example, the "open" function) with the address of this symbol (relative to the start of the libc).

Next, we end our very long header line (with a '\n' character), and start a new header line of nearly 576MB. This new header line is first written to the buffer y, but when y is full, stralloc_readyplus() allocates a new buffer t of roughly 576MB (the size of y plus one eighth), the exact size of the mmap region that we previously munmap()ed:

```
-------|-------|-----------------------------------------|-------+-|----
XXXXXXX|   y   |                       x                 |    t    |ibc
-------|-------|-----------------------------------------|-------+-|----
```

Consequently, we completely control the first pages of the libc (they contain the end of our new header line): we control the .dynsym section, and we replace the address of the "open" function with the address of the "system" function. This method works because Debian's qmail-local binary is partial-RELRO only, and because the open() function has not been called yet, and has therefore not been resolved yet.

Last, we end our new header line, and when qmail-local returns from bouncexf() and calls qmesearch() to open() the ".qmail-extension" file, system(".qmail-extension") is called instead. Because we control this "extension" (it is an extension of the local recipient's mail address, for example localuser-extension@...aldomain), we can execute arbitrary shell commands as any user (except root, and a few system users who do not own their home directory), by sending our large mail message to "localuser-;command;@localdomain".

Last-minute note: the exploitation of glibc's free() to munmap() arbitrary memory regions has been discussed before, in http://tukan.farm/2016/07/27/munmap-madness/.

qmail-verify

CVE-2020-3811

Although the original qmail-smtpd does accept our recipient address "localuser-;command;@localdomain", Debian's qmail-smtpd should not, because it validates the recipient address with an external program qmail-verify (which should reject our recipient address, because the file "~localuser/.qmail-;command;" does not exist). Unfortunately, qmail-verify does reject "localuser-;command;@localdomain", but it accepts the unqualified "localuser-;command;" (without the @localdomain), because:

- it never calls the control_init() function;

- it therefore initializes its default domain to the hard-coded string
  "envnoathost";

- and accepts any unqualified mail address as valid by default (because
  its default domain "envnoathost" is not one of qmail's local domains,
  and is therefore unverifiable).

CVE-2020-3812

We also discovered a minor information disclosure in qmail-verify:
a local attacker can test for the existence of files and directories
anywhere in the filesystem (even in inaccessible directories), because
qmail-verify runs as root and tests for the existence of files in the
attacker's home directory, without dropping its privileges first. For
example (qmail-verify listens on 127.0.0.1:11113 by default):

```
$ ls -l /root/.bashrc
ls: cannot access '/root/.bashrc': Permission denied

$ rm -f ~john/.qmail-test
$ ln -s /root/.bashrc ~john/.qmail-test

$ echo -n 'john-test@...aldomain' | nc -w 2 -u 127.0.0.1 11113 | hexdump -C
00000000  a0 6a 6f 68 6e 2d 74 65  73 74                    |.john-test|
```

The least significant bit of this response's first byte (a0) is 0: the
file "/root/.bashrc" exists.

```
$ ls -l /root/.abcdef
ls: cannot access '/root/.abcdef': Permission denied

$ rm -f ~john/.qmail-test
$ ln -s /root/.abcdef ~john/.qmail-test

$ echo -n 'john-test@...aldomain' | nc -w 2 -u 127.0.0.1 11113 | hexdump -C
00000000  e1 6a 6f 68 6e 2d 74 65  73 74                    |.john-test|
```

The least significant bit of this response's first byte (e1) is 1: the
file "/root/.abcdef" does not exist.

Mitigations

As recommended by Daniel J. Bernstein, qmail can be protected against
all three 2005 CVEs by placing a low, configurable memory limit (a
"softlimit") in the startup scripts of all qmail services.

Alternatively:

qmail can be protected against the RCE (Remote Code Execution) by
configuring the file "control/databytes", which contains the maximum
size of a mail message (this file does not exist by default, and qmail
is therefore remotely exploitable in its default configuration).

Unfortunately, this does not protect qmail against the LPE (Local
Privilege Escalation), because the file "control/databytes" is used
exclusively by qmail-smtpd.

Acknowledgments

We thank Andrew Richards, Alexander Peslyak, the members of
distros@...nwall, and the developers of notqmail for their hard work on
this coordinated release. We also thank Daniel J. Bernstein, and Georgi
Guninski. Finally, we thank Julien Barthelemy, Stephane Bellenger, and
Jean-Paul Michel for their inspiring work.

Patches

We wrote a simple patch for Debian's qmail package (below) that fixes
CVE-2020-3811 and CVE-2020-3812 in qmail-verify, and fixes all three
2005 CVEs in qmail (by hard-coding a safe, upper memory limit in the
alloc() function).

Alternatively:

- an updated version of qmail-verify will be available at
  https://free.acrconsulting.co.uk/email/qmail-verify.html after the
  Coordinated Release Date;

- the developers of notqmail (https://notqmail.org/) have written their
  own patches for the three 2005 CVEs and have started to systematically
  fix all integer overflows and signedness errors in qmail.

```
diff -r -u netqmail_1.06-6/alloc.c netqmail_1.06-6+patches/alloc.c
--- netqmail_1.06-6/alloc.c     1998-06-15 03:53:16.000000000 -0700
+++ netqmail_1.06-6+patches/alloc.c     2020-05-04 16:43:32.923310325 -0700
@@ -1,3 +1,4 @@
+#include <limits.h>
 #include "alloc.h"
 #include "error.h"
 extern char *malloc();
@@ -15,6 +16,10 @@
 unsigned int n;
 {
   char *x;
+  if (n >= (INT_MAX >> 3)) {
+    errno = error_nomem;
+    return 0;
+  }
   n = ALIGNMENT + n - (n & (ALIGNMENT - 1)); /* XXX: could overflow */
   if (n <= avail) { avail -= n; return space + avail; }
   x = malloc(n);
diff -r -u netqmail_1.06-6/qmail-verify.c netqmail_1.06-6+patches/qmail-verify.c
--- netqmail_1.06-6/qmail-verify.c      2020-05-02 09:02:51.954415101 -0700
+++ netqmail_1.06-6+patches/qmail-verify.c      2020-05-08 04:47:27.555539058 -0700
@@ -16,6 +16,8 @@
 #include <sys/types.h>
 #include <sys/stat.h>
 #include <unistd.h>
+#include <limits.h>
+#include <grp.h>
 #include <pwd.h>
 #include <sys/socket.h>
 #include <netinet/in.h>
@@ -38,6 +40,7 @@
 #include "ip.h"
 #include "qmail-verify.h"
 #include "errbits.h"
+#include "scan.h"

 #define enew()  { eout("qmail-verify: "); }
```

```
 #define GETPW_USERLEN 32
@@ -71,6 +74,7 @@
 void die_comms()   { enew(); eout("Misc. comms problem: exiting.\n"); eflush(); _exit(1); }
 void die_inuse()   { enew(); eout("Port already in use: exiting.\n"); eflush(); _exit(1); }
 void die_socket()  { enew(); eout("Error setting up socket: exiting.\n"); eflush(); _exit(1); }
+void die_privs()   { enew(); eout("Unable to drop/restore privileges: exiting.\n"); eflush(); _exit(1); }
 
 char *posstr(buf,status)
 char *buf; int status;
@@ -207,10 +211,47 @@
   return 0;
 }
 
+static int stat_as(uid, gid, path, sbuf)
+const uid_t uid;
+const gid_t gid;
+const char * const path;
+struct stat * const sbuf;
+{
+  static gid_t groups[NGROUPS_MAX + 1];
+  int ngroups = 0;
+  const gid_t saved_egid = getegid();
+  const uid_t saved_euid = geteuid();
+  int ret = -1;
+
+  if (saved_euid == 0) {
+    ngroups = getgroups(sizeof(groups) / sizeof(groups[0]), groups);
+    if (ngroups < 0 ||
+        setgroups(1, &gid) != 0 ||
+        setegid(gid) != 0 ||
+        seteuid(uid) != 0) {
+      die_privs();
+    }
+  }
+
+  ret = stat(path, sbuf);
+
+  if (saved_euid == 0) {
+    if (seteuid(saved_euid) != 0 ||
+        setegid(saved_egid) != 0 ||
+        setgroups(ngroups, groups) != 0) {
+      die_privs();
+    }
+  }
+
+  return ret;
+}
+
 int verifyaddr(addr)
 char *addr;
 {
   char *homedir;
+  uid_t uid = -1;
+  gid_t gid = -1;
   /* static since they get re-used on each call to verifyaddr(). Note
      that they don't need resetting since initial use is always with
      stralloc_copys() except wildchars (reset with ...len=0 below). */
@@ -303,6 +344,7 @@
           if (r == 1)
           {
             char *x;
+            unsigned long u;
             if (!stralloc_ready(&nughde,(unsigned int) dlen)) die_nomem();
             nughde.len = dlen;
             if (cdb_bread(fd,nughde.s,nughde.len) == -1) die_cdb();
@@ -318,10 +360,14 @@
             if (x == nughde.s + nughde.len) return allowaddr(addr,ADDR_OK|QVPOS3);
             ++x;
             /* skip uid */
+            scan_ulong(x,&u);
+            uid = u;
             x += byte_chr(x,nughde.s + nughde.len - x,'\0');
             if (x == nughde.s + nughde.len) return allowaddr(addr,ADDR_OK|QVPOS4);
             ++x;
             /* skip gid */
+            scan_ulong(x,&u);
+            gid = u;
             x += byte_chr(x,nughde.s + nughde.len - x,'\0');
             if (x == nughde.s + nughde.len) return allowaddr(addr,ADDR_OK|QVPOS5);
             ++x;
@@ -360,6 +406,8 @@
   if (!stralloc_copys(&nughde,pw->pw_dir)) die_nomem();
   if (!stralloc_0(&nughde)) die_nomem();
   homedir=nughde.s;
+  uid = pw->pw_uid;
+  gid = pw->pw_gid;
 
   got_nughde:
 
@@ -380,7 +428,7 @@
     if (!stralloc_cat(&qme,&safeext)) die_nomem();
     if (!stralloc_0(&qme)) die_nomem();
 /* e.g. homedir/.qmail-localpart */
-    if (stat(qme.s,&st) == 0) return allowaddr(addr,ADDR_OK|QVPOS10);
+    if (stat_as(uid,gid,qme.s,&st) == 0) return allowaddr(addr,ADDR_OK|QVPOS10);
     if (errno != error_noent) {
       return stat_error(qme.s,errno, STATERR|QVPOS11); /* Maybe not running as root so access denied */
     }
@@ -394,7 +442,7 @@
         if (!stralloc_cats(&qme,"default")) die_nomem();
         if (!stralloc_0(&qme)) die_nomem();
 /* e.g. homedir/.qmail-[xxx-]default */
-        if (stat(qme.s,&st) == 0) {
+        if (stat_as(uid,gid,qme.s,&st) == 0) {
          /* if it's ~alias/.qmail-default, optionally check aliases.cdb */
           if (!i && (quser == auto_usera)) {
             char *s;
@@ -423,6 +471,7 @@
   char *s;
 
   if (chdir(auto_qmail) == -1) die_control();
+  if (control_init() == -1) die_control();
 
   if (control_rldef(&envnoathost,"control/envnoathost",1,"envnoathost") != 1)
     die_control();
```

[https://d1dejaj6dcqv24.cloudfront.net/asset/image/email-banner-384-2x.png]<https://www.qualys.com/email-banner>