python / **cpython** Public

Code Issues 5k+ Pull requests 2.2k Actions Projects Security and q

Commit bca11ae

miss-islington and **serhiy-storchaka** authored on Oct 7, 2025 · 52 / 65 · Verified

```
[3.10] gh-139700: Check consistency of the zip64 end of central directory record
(GH-139702) (GH-139708) (#139714)

Support records with "zip64 extensible data" if there are no bytes
prepended to the ZIP file.

(cherry picked from commit 333d4a6)
(cherry picked from commit 162997b)

Co-authored-by: Serhiy Storchaka <storchaka@gmail.com>
```

3.10 (#139714) · v3.10.20 v3.10.19

1 parent 7317e0b commit bca11ae

3 files changed +113 -23

Top

Filter files…

- Lib
  - test
    - test_zipfile.py
    - zipfile.py
- Misc/NEWS.d/next/Security
  - 2025-10-07-19-31-34.gh-issue-139700.vNHU1O.rst

Search within code

Lib/test/test_zipfile.py

```
@@ -862,6 +862,8 @@ def make_zip64_file(
```

```
862  862              self, file_size_64_set=False, file_size_extra=False,
863  863              compress_size_64_set=False, compress_size_extra=False,
864  864              header_offset_64_set=False, header_offset_extra=False,
     865  +           extensible_data=b'',
     866  +           end_of_central_dir_size=None, offset_to_end_of_central_dir=None,
865  867          ):
866  868              """Generate bytes sequence for a zip with (incomplete) zip64 data.
867  869
     @@ -915,6 +917,12 @@ def make_zip64_file(
915  917
916  918              central_dir_size = struct.pack('<Q', 58 + 8 *
     len(central_zip64_fields))
917  919              offset_to_central_dir = struct.pack('<Q', 50 + 8 *
     len(local_zip64_fields))
     920  +           if end_of_central_dir_size is None:
     921  +               end_of_central_dir_size = 44 + len(extensible_data)
     922  +           if offset_to_end_of_central_dir is None:
     923  +               offset_to_end_of_central_dir = (108
     924  +                                               + 8 * len(local_zip64_fields)
     925  +                                               + 8 * len(central_zip64_fields))
918  926
919  927              local_extra_length = struct.pack("<H", 4 + 8 *
     len(local_zip64_fields))
920  928              central_extra_length = struct.pack("<H", 4 + 8 *
     len(central_zip64_fields))
     @@ -943,14 +951,17 @@ def make_zip64_file(
943  951                  + filename
944  952                  + central_extra
945  953                  # Zip64 end of central directory
946      -           + b"PK\x06\x06,\x00\x00\x00\x00\x00\x00\x00-\x00-"
947      -           + b"\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
     954  +           + b"PK\x06\x06"
     955  +           + struct.pack('<Q', end_of_central_dir_size)
     956  +           + b"-\x00-
     \x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
948  957                  + b"\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
949  958                  + central_dir_size
950  959                  + offset_to_central_dir
```

```
                960 +             + extensible_data
951             961               # Zip64 end of central directory
952             962 -             + b"PK\x06\x07\x00\x00\x00\x00l\x00\x00\x00\x00\x00\x00\x00\x01"
953             963 -             + b"\x00\x00\x00"
                964 +             + b"PK\x06\x07\x00\x00\x00\x00"
                965 +             + struct.pack('<Q', offset_to_end_of_central_dir)
                966 +             + b"\x01\x00\x00\x00"
954             967               # end of central directory
955             968               + b"PK\x05\x06\x00\x00\x00\x00\x01\x00\x01\x00:\x00\x00\x002\x00"
956             969               + b"\x00\x00\x00\x00"
                    @@ -981,6 +992,7 @@ def test_bad_zip64_extra(self):
981             992           with self.assertRaises(zipfile.BadZipFile) as e:
982             993               zipfile.ZipFile(io.BytesIO(missing_file_size_extra))
983             994           self.assertIn('file size', str(e.exception).lower())
                995 +
                    self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_file_size_extra)))
984             996
985             997           # zip64 file size present, zip64 compress size present, one field in
986             998           # extra, expecting two, equals missing compress size.
                    @@ -992,6 +1004,7 @@ def test_bad_zip64_extra(self):
992            1004           with self.assertRaises(zipfile.BadZipFile) as e:
993            1005               zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
994            1006           self.assertIn('compress size', str(e.exception).lower())
               1007 +
                    self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
995            1008
996            1009           # zip64 compress size present, no fields in extra, expecting one,
997            1010           # equals missing compress size.
                    @@ -1001,6 +1014,7 @@ def test_bad_zip64_extra(self):
1001           1014           with self.assertRaises(zipfile.BadZipFile) as e:
1002           1015               zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1003           1016           self.assertIn('compress size', str(e.exception).lower())
               1017 +
                    self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1004           1018
1005           1019           # zip64 file size present, zip64 compress size present, zip64 header
1006           1020           # offset present, two fields in extra, expecting three, equals
                    missing
                    @@ -1015,6 +1029,7 @@ def test_bad_zip64_extra(self):
```

```
1015 1029             with self.assertRaises(zipfile.BadZipFile) as e:
1016 1030                 zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1017 1031             self.assertIn('header offset', str(e.exception).lower())
     1032 +
                   self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1018 1033
1019 1034             # zip64 compress size present, zip64 header offset present, one field
1020 1035             # in extra, expecting two, equals missing header offset
          @@ -1027,6 +1042,7 @@ def test_bad_zip64_extra(self):
1027 1042             with self.assertRaises(zipfile.BadZipFile) as e:
1028 1043                 zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1029 1044             self.assertIn('header offset', str(e.exception).lower())
     1045 +
                   self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1030 1046
1031 1047             # zip64 file size present, zip64 header offset present, one field in
1032 1048             # extra, expecting two, equals missing header offset
          @@ -1039,6 +1055,7 @@ def test_bad_zip64_extra(self):
1039 1055             with self.assertRaises(zipfile.BadZipFile) as e:
1040 1056                 zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1041 1057             self.assertIn('header offset', str(e.exception).lower())
     1058 +
                   self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1042 1059
1043 1060             # zip64 header offset present, no fields in extra, expecting one,
1044 1061             # equals missing header offset
          @@ -1050,6 +1067,63 @@ def test_bad_zip64_extra(self):
1050 1067             with self.assertRaises(zipfile.BadZipFile) as e:
1051 1068                 zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1052 1069             self.assertIn('header offset', str(e.exception).lower())
     1070 +
                   self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
     1071 +
     1072 +     def test_bad_zip64_end_of_central_dir(self):
     1073 +         zipdata = self.make_zip64_file(end_of_central_dir_size=0)
     1074 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
     1075 +             zipfile.ZipFile(io.BytesIO(zipdata))
     1076 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
     1077 +
     1078 +         zipdata = self.make_zip64_file(end_of_central_dir_size=100)
```

```
1079 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1080 +             zipfile.ZipFile(io.BytesIO(zipdata))
1081 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1082 +
1083 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=0)
1084 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1085 +             zipfile.ZipFile(io.BytesIO(zipdata))
1086 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1087 +
1088 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=1000)
1089 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*locator'):
1090 +             zipfile.ZipFile(io.BytesIO(zipdata))
1091 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1092 +
1093 +     def test_zip64_end_of_central_dir_record_not_found(self):
1094 +         zipdata = self.make_zip64_file()
1095 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1096 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1097 +             zipfile.ZipFile(io.BytesIO(zipdata))
1098 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1099 +
1100 +         zipdata = self.make_zip64_file(
1101 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1102 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1103 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1104 +             zipfile.ZipFile(io.BytesIO(zipdata))
1105 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1106 +
1107 +     def test_zip64_extensible_data(self):
1108 +         # These values are what is set in the make_zip64_file method.
1109 +         expected_file_size = 8
1110 +         expected_compress_size = 8
1111 +         expected_header_offset = 0
1112 +         expected_content = b"test1234"
1113 +
1114 +         zipdata = self.make_zip64_file(
1115 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1116 +         with zipfile.ZipFile(io.BytesIO(zipdata)) as zf:
1117 +             zinfo = zf.infolist()[0]
1118 +             self.assertEqual(zinfo.file_size, expected_file_size)
```

```
        1119 +             self.assertEqual(zinfo.compress_size, expected_compress_size)
        1120 +             self.assertEqual(zinfo.header_offset, expected_header_offset)
        1121 +             self.assertEqual(zf.read(zinfo), expected_content)
        1122 +         self.assertTrue(zipfile.is_zipfile(io.BytesIO(zipdata)))
        1123 +
        1124 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
        1125 +             zipfile.ZipFile(io.BytesIO(b'prepended' + zipdata))
        1126 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(b'prepended' +
             zipdata)))
1053    1127
1054    1128         def test_generated_valid_zip64_extra(self):
1055    1129             # These values are what is set in the make_zip64_file method.
```

Lib/zipfile.py

```
             @@ -210,41 +210,57 @@ def is_zipfile(filename):
210     210              else:
211     211                  with open(filename, "rb") as fp:
212     212                      result = _check_zipfile(fp)
213         -     except OSError:
        213 +     except (OSError, BadZipFile):
214     214              pass
215     215          return result
216     216
217     217  def _EndRecData64(fpin, offset, endrec):
218     218          """
219     219          Read the ZIP64 end-of-archive records and use that to update endrec
220     220          """
221         -     try:
222         -         fpin.seek(offset - sizeEndCentDir64Locator, 2)
223         -     except OSError:
224         -         # If the seek fails, the file is not large enough to contain a ZIP64
        221 +     offset -= sizeEndCentDir64Locator
        222 +     if offset < 0:
        223 +         # The file is not large enough to contain a ZIP64
225     224              # end-of-archive record, so just return the end record we were given.
226     225              return endrec
227         -
        226 +     fpin.seek(offset)
228     227          data = fpin.read(sizeEndCentDir64Locator)
```

```
229  228          if len(data) != sizeEndCentDir64Locator:
230       -           return endrec
     229  +           raise OSError("Unknown I/O error")
231  230          sig, diskno, reloff, disks = struct.unpack(structEndArchive64Locator,
                 data)
232  231          if sig != stringEndArchive64Locator:
233  232              return endrec
234  233
235  234          if diskno != 0 or disks > 1:
236  235              raise BadZipFile("zipfiles that span multiple disks are not
                 supported")
237  236
238       -      # Assume no 'zip64 extensible data'
239       -      fpin.seek(offset - sizeEndCentDir64Locator - sizeEndCentDir64, 2)
     237  +      offset -= sizeEndCentDir64
     238  +      if reloff > offset:
     239  +          raise BadZipFile("Corrupt zip64 end of central directory locator")
     240  +      # First, check the assumption that there is no prepended data.
     241  +      fpin.seek(reloff)
     242  +      extrasz = offset - reloff
240  243          data = fpin.read(sizeEndCentDir64)
241  244          if len(data) != sizeEndCentDir64:
242       -           return endrec
     245  +           raise OSError("Unknown I/O error")
     246  +      if not data.startswith(stringEndArchive64) and reloff != offset:
     247  +          # Since we already have seen the Zip64 EOCD Locator, it's
     248  +          # possible we got here because there is prepended data.
     249  +          # Assume no 'zip64 extensible data'
     250  +          fpin.seek(offset)
     251  +          extrasz = 0
     252  +          data = fpin.read(sizeEndCentDir64)
     253  +          if len(data) != sizeEndCentDir64:
     254  +              raise OSError("Unknown I/O error")
     255  +      if not data.startswith(stringEndArchive64):
     256  +          raise BadZipFile("Zip64 end of central directory record not found")
     257  +
243  258          sig, sz, create_version, read_version, disk_num, disk_dir, \
244  259              dircount, dircount2, dirsize, diroffset = \
245  260              struct.unpack(structEndArchive64, data)
246       -      if sig != stringEndArchive64:
```

```
247      -          return endrec
         261 +      if (diroffset + dirsize != reloff or
         262 +          sz + 12 != sizeEndCentDir64 + extrasz):
         263 +          raise BadZipFile("Corrupt zip64 end of central directory record")
248      264
249      265        # Update the original endrec using data from the ZIP64 record
250      266        endrec[_ECD_SIGNATURE] = sig
         @@ -254,6 +270,7 @@ def _EndRecData64(fpin, offset, endrec):
254      270        endrec[_ECD_ENTRIES_TOTAL] = dircount2
255      271        endrec[_ECD_SIZE] = dirsize
256      272        endrec[_ECD_OFFSET] = diroffset
         273 +      endrec[_ECD_LOCATION] = offset - extrasz
257      274        return endrec
258      275
259      276
         @@ -287,7 +304,7 @@ def _EndRecData(fpin):
287      304            endrec.append(filesize - sizeEndCentDir)
288      305
289      306            # Try to read the "Zip64 end of central directory" structure
290      -          return _EndRecData64(fpin, -sizeEndCentDir, endrec)
         307 +          return _EndRecData64(fpin, filesize - sizeEndCentDir, endrec)
291      308
292      309        # Either this is not a ZIP file, or it is a ZIP file with an archive
293      310        # comment.  Search the end of the file for the "end of central directory"
         @@ -311,8 +328,7 @@ def _EndRecData(fpin):
311      328            endrec.append(maxCommentStart + start)
312      329
313      330            # Try to read the "Zip64 end of central directory" structure
314      -          return _EndRecData64(fpin, maxCommentStart + start - filesize,
315      -                               endrec)
         331 +          return _EndRecData64(fpin, maxCommentStart + start, endrec)
316      332
317      333        # Unable to find a valid end of central directory structure
318      334        return None
         @@ -1345,9 +1361,6 @@ def _RealGetContents(self):
1345     1361
1346     1362            # "concat" is zero, unless zip was concatenated to another file
1347     1363            concat = endrec[_ECD_LOCATION] - size_cd - offset_cd
```

```
1348        -         if endrec[_ECD_SIGNATURE] == stringEndArchive64:
1349        -             # If Zip64 extension structures are present, account for them
1350        -             concat -= (sizeEndCentDir64 + sizeEndCentDir64Locator)
1351  1364
1352  1365            if self.debug > 2:
1353  1366                inferred = concat + offset_cd
      ↓ ↑   @@ -1928,7 +1941,7 @@ def _write_end_record(self):
1928  1941                                        " would require ZIP64 extensions")
1929  1942                zip64endrec = struct.pack(
1930  1943                    structEndArchive64, stringEndArchive64,
1931        -                   44, 45, 45, 0, 0, centDirCount, centDirCount,
      1944  +                   sizeEndCentDir64 - 12, 45, 45, 0, 0, centDirCount,
          centDirCount,
1932  1945                    centDirSize, centDirOffset)
1933  1946                self.fp.write(zip64endrec)
1934  1947
      ↓
```

...5-10-07-19-31-34.gh-issue-139700.vNHU1O.rst

```
•••   @@ -0,0 +1,3 @@
1   + Check consistency of the zip64 end of central directory record. Support
2   + records with "zip64 extensible data" if there are no bytes prepended to the
3   + ZIP file.
```

Comments 0