python / **cpython** Public

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

⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Commit 76437ac

miss-islington and **serhiy-storchaka** authored on Oct 7, 2025 · ✕ 52 / 69 · Verified

```
[3.9] gh-139700: Check consistency of the zip64 end of central directory record
(GH-139702) (GH-139708) (#139715)

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>
```

· v3.9.25 ••• 3.9

1 parent e26ba93 commit 76437ac

3 files changed +113 -23 lines changed

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

3 files changed +113 -23 lines changed

Search within code

Lib/test/test_zipfile.py

```
@@ -859,6 +859,8 @@ def make_zip64_file(
            self, file_size_64_set=False, file_size_extra=False,
            compress_size_64_set=False, compress_size_extra=False,
            header_offset_64_set=False, header_offset_extra=False,
+           extensible_data=b'',
+           end_of_central_dir_size=None, offset_to_end_of_central_dir=None,
        ):
            """Generate bytes sequence for a zip with (incomplete) zip64 data.

@@ -912,6 +914,12 @@ def make_zip64_file(

            central_dir_size = struct.pack('<Q', 58 + 8 * len(central_zip64_fields))
            offset_to_central_dir = struct.pack('<Q', 50 + 8 * len(local_zip64_fields))
+           if end_of_central_dir_size is None:
+               end_of_central_dir_size = 44 + len(extensible_data)
+           if offset_to_end_of_central_dir is None:
+               offset_to_end_of_central_dir = (108
+                                               + 8 * len(local_zip64_fields)
+                                               + 8 * len(central_zip64_fields))

            local_extra_length = struct.pack("<H", 4 + 8 * len(local_zip64_fields))
            central_extra_length = struct.pack("<H", 4 + 8 * len(central_zip64_fields))
@@ -940,14 +948,17 @@ def make_zip64_file(
                + filename
                + central_extra
                # Zip64 end of central directory
-               + b"PK\x06\x06,\x00\x00\x00\x00\x00\x00\x00-\x00-"
-               + b"\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
+               + b"PK\x06\x06"
+               + struct.pack('<Q', end_of_central_dir_size)
+               + b"-\x00-\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
                + b"\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
```

```
946  955                 + central_dir_size
947  956                 + offset_to_central_dir
     957  +              + extensible_data
948  958                 # Zip64 end of central directory locator
949       -              + b"PK\x06\x07\x00\x00\x00\x00l\x00\x00\x00\x00\x00\x00\x00\x01"
950       -              + b"\x00\x00\x00"
     959  +              + b"PK\x06\x07\x00\x00\x00\x00"
     960  +              + struct.pack('<Q', offset_to_end_of_central_dir)
     961  +              + b"\x01\x00\x00\x00"
951  962                 # end of central directory
952  963                 + b"PK\x05\x06\x00\x00\x00\x00\x01\x00\x01\x00:\x00\x00\x002\x00"
953  964                 + b"\x00\x00\x00\x00"
          @@ -978,6 +989,7 @@ def test_bad_zip64_extra(self):
978  989             with self.assertRaises(zipfile.BadZipFile) as e:
979  990                 zipfile.ZipFile(io.BytesIO(missing_file_size_extra))
980  991             self.assertIn('file size', str(e.exception).lower())
     992  +
          self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_file_size_extra)))
981  993
982  994             # zip64 file size present, zip64 compress size present, one field in
983  995             # extra, expecting two, equals missing compress size.
          @@ -989,6 +1001,7 @@ def test_bad_zip64_extra(self):
989  1001            with self.assertRaises(zipfile.BadZipFile) as e:
990  1002                zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
991  1003            self.assertIn('compress size', str(e.exception).lower())
     1004 +
          self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
992  1005
993  1006            # zip64 compress size present, no fields in extra, expecting one,
994  1007            # equals missing compress size.
          @@ -998,6 +1011,7 @@ def test_bad_zip64_extra(self):
998  1011            with self.assertRaises(zipfile.BadZipFile) as e:
999  1012                zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1000 1013            self.assertIn('compress size', str(e.exception).lower())
     1014 +
          self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1001 1015
1002 1016            # zip64 file size present, zip64 compress size present, zip64 header
```

```
1003  1017            # offset present, two fields in extra, expecting three, equals
                      missing
      @@ -1012,6 +1026,7 @@ def test_bad_zip64_extra(self):
1012  1026            with self.assertRaises(zipfile.BadZipFile) as e:
1013  1027                zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1014  1028            self.assertIn('header offset', str(e.exception).lower())
      1029  +
                      self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1015  1030
1016  1031            # zip64 compress size present, zip64 header offset present, one field
1017  1032            # in extra, expecting two, equals missing header offset
      @@ -1024,6 +1039,7 @@ def test_bad_zip64_extra(self):
1024  1039            with self.assertRaises(zipfile.BadZipFile) as e:
1025  1040                zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1026  1041            self.assertIn('header offset', str(e.exception).lower())
      1042  +
                      self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1027  1043
1028  1044            # zip64 file size present, zip64 header offset present, one field in
1029  1045            # extra, expecting two, equals missing header offset
      @@ -1036,6 +1052,7 @@ def test_bad_zip64_extra(self):
1036  1052            with self.assertRaises(zipfile.BadZipFile) as e:
1037  1053                zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1038  1054            self.assertIn('header offset', str(e.exception).lower())
      1055  +
                      self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1039  1056
1040  1057            # zip64 header offset present, no fields in extra, expecting one,
1041  1058            # equals missing header offset
      @@ -1047,6 +1064,63 @@ def test_bad_zip64_extra(self):
1047  1064            with self.assertRaises(zipfile.BadZipFile) as e:
1048  1065                zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1049  1066            self.assertIn('header offset', str(e.exception).lower())
      1067  +
                      self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
      1068  +
      1069  +     def test_bad_zip64_end_of_central_dir(self):
      1070  +         zipdata = self.make_zip64_file(end_of_central_dir_size=0)
      1071  +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
      1072  +             zipfile.ZipFile(io.BytesIO(zipdata))
```

```
1073 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1074 +
1075 +         zipdata = self.make_zip64_file(end_of_central_dir_size=100)
1076 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1077 +             zipfile.ZipFile(io.BytesIO(zipdata))
1078 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1079 +
1080 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=0)
1081 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1082 +             zipfile.ZipFile(io.BytesIO(zipdata))
1083 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1084 +
1085 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=1000)
1086 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*locator'):
1087 +             zipfile.ZipFile(io.BytesIO(zipdata))
1088 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1089 +
1090 +     def test_zip64_end_of_central_dir_record_not_found(self):
1091 +         zipdata = self.make_zip64_file()
1092 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1093 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1094 +             zipfile.ZipFile(io.BytesIO(zipdata))
1095 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1096 +
1097 +         zipdata = self.make_zip64_file(
1098 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1099 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1100 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1101 +             zipfile.ZipFile(io.BytesIO(zipdata))
1102 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1103 +
1104 +     def test_zip64_extensible_data(self):
1105 +         # These values are what is set in the make_zip64_file method.
1106 +         expected_file_size = 8
1107 +         expected_compress_size = 8
1108 +         expected_header_offset = 0
1109 +         expected_content = b"test1234"
1110 +
1111 +         zipdata = self.make_zip64_file(
1112 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
```

```
        1113 +         with zipfile.ZipFile(io.BytesIO(zipdata)) as zf:
        1114 +             zinfo = zf.infolist()[0]
        1115 +             self.assertEqual(zinfo.file_size, expected_file_size)
        1116 +             self.assertEqual(zinfo.compress_size, expected_compress_size)
        1117 +             self.assertEqual(zinfo.header_offset, expected_header_offset)
        1118 +             self.assertEqual(zf.read(zinfo), expected_content)
        1119 +         self.assertTrue(zipfile.is_zipfile(io.BytesIO(zipdata)))
        1120 +
        1121 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
        1122 +             zipfile.ZipFile(io.BytesIO(b'prepended' + zipdata))
        1123 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(b'prepended' +
             zipdata)))
1050    1124
1051    1125         def test_generated_valid_zip64_extra(self):
1052    1126             # These values are what is set in the make_zip64_file method.
```

Lib/zipfile.py

```
             @@ -206,41 +206,57 @@ def is_zipfile(filename):
206     206             else:
207     207                 with open(filename, "rb") as fp:
208     208                     result = _check_zipfile(fp)
209         -     except OSError:
        209 +     except (OSError, BadZipFile):
210     210             pass
211     211         return result
212     212
213     213     def _EndRecData64(fpin, offset, endrec):
214     214         """
215     215         Read the ZIP64 end-of-archive records and use that to update endrec
216     216         """
217         -     try:
218         -         fpin.seek(offset - sizeEndCentDir64Locator, 2)
219         -     except OSError:
220         -         # If the seek fails, the file is not large enough to contain a ZIP64
        217 +     offset -= sizeEndCentDir64Locator
        218 +     if offset < 0:
        219 +         # The file is not large enough to contain a ZIP64
221     220             # end-of-archive record, so just return the end record we were given.
222     221             return endrec
```

```
223       -
          222 +     fpin.seek(offset)
224       223         data = fpin.read(sizeEndCentDir64Locator)
225       224         if len(data) != sizeEndCentDir64Locator:
226       -             return endrec
          225 +             raise OSError("Unknown I/O error")
227       226         sig, diskno, reloff, disks = struct.unpack(structEndArchive64Locator,
                    data)
228       227         if sig != stringEndArchive64Locator:
229       228             return endrec
230       229
231       230         if diskno != 0 or disks > 1:
232       231             raise BadZipFile("zipfiles that span multiple disks are not
                    supported")
233       232
234       -     # Assume no 'zip64 extensible data'
235       -     fpin.seek(offset - sizeEndCentDir64Locator - sizeEndCentDir64, 2)
          233 +     offset -= sizeEndCentDir64
          234 +     if reloff > offset:
          235 +         raise BadZipFile("Corrupt zip64 end of central directory locator")
          236 +     # First, check the assumption that there is no prepended data.
          237 +     fpin.seek(reloff)
          238 +     extrasz = offset - reloff
236       239         data = fpin.read(sizeEndCentDir64)
237       240         if len(data) != sizeEndCentDir64:
238       -             return endrec
          241 +             raise OSError("Unknown I/O error")
          242 +     if not data.startswith(stringEndArchive64) and reloff != offset:
          243 +         # Since we already have seen the Zip64 EOCD Locator, it's
          244 +         # possible we got here because there is prepended data.
          245 +         # Assume no 'zip64 extensible data'
          246 +         fpin.seek(offset)
          247 +         extrasz = 0
          248 +         data = fpin.read(sizeEndCentDir64)
          249 +         if len(data) != sizeEndCentDir64:
          250 +             raise OSError("Unknown I/O error")
          251 +     if not data.startswith(stringEndArchive64):
          252 +         raise BadZipFile("Zip64 end of central directory record not found")
          253 +
239       254         sig, sz, create_version, read_version, disk_num, disk_dir, \
```

```
240  255            dircount, dircount2, dirsize, diroffset = \
241  256            struct.unpack(structEndArchive64, data)
242      -     if sig != stringEndArchive64:
243      -         return endrec
     257  +     if (diroffset + dirsize != reloff or
     258  +         sz + 12 != sizeEndCentDir64 + extrasz):
     259  +         raise BadZipFile("Corrupt zip64 end of central directory record")
244  260
245  261        # Update the original endrec using data from the ZIP64 record
246  262        endrec[_ECD_SIGNATURE] = sig
          @@ -250,6 +266,7 @@ def _EndRecData64(fpin, offset, endrec):
250  266        endrec[_ECD_ENTRIES_TOTAL] = dircount2
251  267        endrec[_ECD_SIZE] = dirsize
252  268        endrec[_ECD_OFFSET] = diroffset
     269  +     endrec[_ECD_LOCATION] = offset - extrasz
253  270        return endrec
254  271
255  272
          @@ -283,7 +300,7 @@ def _EndRecData(fpin):
283  300            endrec.append(filesize - sizeEndCentDir)
284  301
285  302            # Try to read the "Zip64 end of central directory" structure
286      -         return _EndRecData64(fpin, -sizeEndCentDir, endrec)
     303  +         return _EndRecData64(fpin, filesize - sizeEndCentDir, endrec)
287  304
288  305        # Either this is not a ZIP file, or it is a ZIP file with an archive
289  306        # comment.  Search the end of the file for the "end of central directory"
          @@ -307,8 +324,7 @@ def _EndRecData(fpin):
307  324            endrec.append(maxCommentStart + start)
308  325
309  326            # Try to read the "Zip64 end of central directory" structure
310      -         return _EndRecData64(fpin, maxCommentStart + start - filesize,
311      -                              endrec)
     327  +         return _EndRecData64(fpin, maxCommentStart + start, endrec)
312  328
313  329        # Unable to find a valid end of central directory structure
314  330        return None
          @@ -1341,9 +1357,6 @@ def _RealGetContents(self):
```

```
1341  1357
1342  1358            # "concat" is zero, unless zip was concatenated to another file
1343  1359            concat = endrec[_ECD_LOCATION] - size_cd - offset_cd
1344        -         if endrec[_ECD_SIGNATURE] == stringEndArchive64:
1345        -             # If Zip64 extension structures are present, account for them
1346        -             concat -= (sizeEndCentDir64 + sizeEndCentDir64Locator)
1347  1360
1348  1361            if self.debug > 2:
1349  1362                inferred = concat + offset_cd
      ↓↑    @@ -1922,7 +1935,7 @@ def _write_end_record(self):
1922  1935                                        " would require ZIP64 extensions")
1923  1936                zip64endrec = struct.pack(
1924  1937                    structEndArchive64, stringEndArchive64,
1925        -                   44, 45, 45, 0, 0, centDirCount, centDirCount,
      1938  +                   sizeEndCentDir64 - 12, 45, 45, 0, 0, centDirCount,
          centDirCount,
1926  1939                    centDirSize, centDirOffset)
1927  1940                self.fp.write(zip64endrec)
1928  1941
      ↓
```

...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