

python / cpython Public

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

# Commit d11e69d



miss-islington and serhiy-storchaka authored on Oct 7, 2025 · ✖ 55 / 59 · Verified

[3.14] [gh-139700](#): Check consistency of the zip64 end of central directory record ([GH-139702](#)) ([GH-139706](#))

Support records with "zip64 extensible data" if there are no bytes prepended to the ZIP file.  
(cherry picked from commit [162997b](#))

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

[3.14](#) (#139706) · [v3.14.4](#) ... [v3.14.1](#)

1 parent [09fdd21](#) commit d11e69d

**3 files changed** +113 -23 lines changed ↑ Top

Filter files...

- ✓ Lib
  - ✓ test/test\_zipfile
    - test\_core.py
  - ✓ zipfile
    - \_\_init\_\_.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/test_core.py
  ↑↑ -898,6 +898,8 @@ def make_zip64_file(
  898      898          self, file_size_64_set=False, file_size_extra=False,

```

```

899 899         compress_size_64_set=False, compress_size_extra=False,
900 900         header_offset_64_set=False, header_offset_extra=False,
901 901     +         extensible_data=b'',
902 902     +         end_of_central_dir_size=None, offset_to_end_of_central_dir=None,
901 903     ):
902 904         """Generate bytes sequence for a zip with (incomplete) zip64 data.
903 905
904 906     @@ -951,6 +953,12 @@ def make_zip64_file(
905 907
906 908         central_dir_size = struct.pack('<Q', 58 + 8 *
907 909         len(central_zip64_fields))
908 910         offset_to_central_dir = struct.pack('<Q', 50 + 8 *
909 911         len(local_zip64_fields))
910 912     +         if end_of_central_dir_size is None:
911 913     +             end_of_central_dir_size = 44 + len(extensible_data)
912 914     +         if offset_to_end_of_central_dir is None:
913 915     +             offset_to_end_of_central_dir = (108
914 916     +                 + 8 * len(local_zip64_fields)
915 917     +                 + 8 * len(central_zip64_fields))
916 918
917 919         local_extra_length = struct.pack("<H", 4 + 8 *
918 920         len(local_zip64_fields))
919 921         central_extra_length = struct.pack("<H", 4 + 8 *
920 922         len(central_zip64_fields))
921 923     @@ -979,14 +987,17 @@ def make_zip64_file(
922 924
923 925         + filename
924 926         + central_extra
925 927         # Zip64 end of central directory
926 928     + b"PK\x06\x06,\x00\x00\x00\x00\x00\x00\x00-\x00-"
927 929     + b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00"
928 930     + b"PK\x06\x06"
929 931     + struct.pack('<Q', end_of_central_dir_size)
930 932     + b"-\x00-
931 933         \x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00"
932 934     + b"\x00\x00\x01\x00\x00\x00\x00\x00\x00"
933 935     + central_dir_size
934 936     + offset_to_central_dir
935 937     + extensible_data

```

```

987  997          # Zip64 end of central directory locator
988  -          + b"PK\x06\x07\x00\x00\x00\x001\x00\x00\x00\x00\x00\x00\x01"
989  -          + b"\x00\x00\x00"
          998  +          + b"PK\x06\x07\x00\x00\x00\x00"
          999  +          + struct.pack('<Q', offset_to_end_of_central_dir)
          1000 +          + b"\x01\x00\x00\x00"
990  1001         # end of central directory
991  1002         + b"PK\x05\x06\x00\x00\x00\x00\x01\x00\x01\x00:\x00\x00\x002\x00"
992  1003         + b"\x00\x00\x00\x00"
          ⚡
          ⚡
          @@ -1017,6 +1028,7 @@ def test_bad_zip64_extra(self):
1017  1028         with self.assertRaises(zipfile.BadZipFile) as e:
1018  1029             zipfile.ZipFile(io.BytesIO(missing_file_size_extra))
1019  1030             self.assertIn('file size', str(e.exception).lower())
          1031  +
          self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_file_size_extra)))
1020  1032
1021  1033         # zip64 file size present, zip64 compress size present, one field in
1022  1034         # extra, expecting two, equals missing compress size.
          ⚡
          @@ -1028,6 +1040,7 @@ def test_bad_zip64_extra(self):
1028  1040         with self.assertRaises(zipfile.BadZipFile) as e:
1029  1041             zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1030  1042             self.assertIn('compress size', str(e.exception).lower())
          1043  +
          self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1031  1044
1032  1045         # zip64 compress size present, no fields in extra, expecting one,
1033  1046         # equals missing compress size.
          ⚡
          @@ -1037,6 +1050,7 @@ def test_bad_zip64_extra(self):
1037  1050         with self.assertRaises(zipfile.BadZipFile) as e:
1038  1051             zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1039  1052             self.assertIn('compress size', str(e.exception).lower())
          1053  +
          self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1040  1054
1041  1055         # zip64 file size present, zip64 compress size present, zip64 header
1042  1056         # offset present, two fields in extra, expecting three, equals
          missing
          ⚡
          @@ -1051,6 +1065,7 @@ def test_bad_zip64_extra(self):
1051  1065         with self.assertRaises(zipfile.BadZipFile) as e:

```

```

1052 1066         zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1053 1067         self.assertIn('header offset', str(e.exception).lower())
1068 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1054 1069
1055 1070         # zip64 compress size present, zip64 header offset present, one field
1056 1071         # in extra, expecting two, equals missing header offset
@@ -1063,6 +1078,7 @@ def test_bad_zip64_extra(self):
1063 1078         with self.assertRaises(zipfile.BadZipFile) as e:
1064 1079             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1065 1080             self.assertIn('header offset', str(e.exception).lower())
1081 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1066 1082
1067 1083         # zip64 file size present, zip64 header offset present, one field in
1068 1084         # extra, expecting two, equals missing header offset
@@ -1075,6 +1091,7 @@ def test_bad_zip64_extra(self):
1075 1091         with self.assertRaises(zipfile.BadZipFile) as e:
1076 1092             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1077 1093             self.assertIn('header offset', str(e.exception).lower())
1094 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1078 1095
1079 1096         # zip64 header offset present, no fields in extra, expecting one,
1080 1097         # equals missing header offset
@@ -1086,6 +1103,63 @@ def test_bad_zip64_extra(self):
1086 1103         with self.assertRaises(zipfile.BadZipFile) as e:
1087 1104             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1088 1105             self.assertIn('header offset', str(e.exception).lower())
1106 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1107 +
1108 +         def test_bad_zip64_end_of_central_dir(self):
1109 +             zipdata = self.make_zip64_file(end_of_central_dir_size=0)
1110 +             with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1111 +                 zipfile.ZipFile(io.BytesIO(zipdata))
1112 +             self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1113 +
1114 +             zipdata = self.make_zip64_file(end_of_central_dir_size=100)
1115 +             with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):

```

```
1116 +         zipfile.ZipFile(io.BytesIO(zipdata))
1117 +     self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1118 +
1119 +     zipdata = self.make_zip64_file(offset_to_end_of_central_dir=0)
1120 +     with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1121 +         zipfile.ZipFile(io.BytesIO(zipdata))
1122 +     self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1123 +
1124 +     zipdata = self.make_zip64_file(offset_to_end_of_central_dir=1000)
1125 +     with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*locator'):
1126 +         zipfile.ZipFile(io.BytesIO(zipdata))
1127 +     self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1128 +
1129 +     def test_zip64_end_of_central_dir_record_not_found(self):
1130 +         zipdata = self.make_zip64_file()
1131 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1132 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1133 +             zipfile.ZipFile(io.BytesIO(zipdata))
1134 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1135 +
1136 +         zipdata = self.make_zip64_file(
1137 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1138 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1139 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1140 +             zipfile.ZipFile(io.BytesIO(zipdata))
1141 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1142 +
1143 +     def test_zip64_extensible_data(self):
1144 +         # These values are what is set in the make_zip64_file method.
1145 +         expected_file_size = 8
1146 +         expected_compress_size = 8
1147 +         expected_header_offset = 0
1148 +         expected_content = b"test1234"
1149 +
1150 +         zipdata = self.make_zip64_file(
1151 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1152 +         with zipfile.ZipFile(io.BytesIO(zipdata)) as zf:
1153 +             zinfo = zf.infolist()[0]
1154 +             self.assertEqual(zinfo.file_size, expected_file_size)
1155 +             self.assertEqual(zinfo.compress_size, expected_compress_size)
```

```

1156 +         self.assertEqual(zinfo.header_offset, expected_header_offset)
1157 +         self.assertEqual(zf.read(zinfo), expected_content)
1158 +         self.assertTrue(zipfile.is_zipfile(io.BytesIO(zipdata)))
1159 +
1160 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1161 +             zipfile.ZipFile(io.BytesIO(b'prepending' + zipdata))
1162 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(b'prepending' +
zipdata)))
1089 1163
1090 1164         def test_generated_valid_zip64_extra(self):
1091 1165             # These values are what is set in the make_zip64_file method.

```

```

Lib/zipfile/__init__.py
↑ ... @@ -265,7 +265,7 @@ def is_zipfile(filename):
265 265         else:
266 266             with open(filename, "rb") as fp:
267 267                 result = _check_zipfile(fp)
268 -         except OSError:
268 +         except (OSError, BadZipFile):
269 269             pass
270 270         return result
271 271
⇅ @@ -275,9 +275,6 @@ def _handle_prepending_data(endrec, debug=0):
275 275
276 276         # "concat" is zero, unless zip was concatenated to another file
277 277         concat = endrec[_ECD_LOCATION] - size_cd - offset_cd
278 -         if endrec[_ECD_SIGNATURE] == stringEndArchive64:
279 -             # If Zip64 extension structures are present, account for them
280 -             concat -= (sizeEndCentDir64 + sizeEndCentDir64Locator)
281 278
282 279         if debug > 2:
283 280             inferred = concat + offset_cd
⇅ @@ -289,33 +286,49 @@ def _EndRecData64(fpin, offset, endrec):
289 286         """
290 287         Read the ZIP64 end-of-archive records and use that to update endrec
291 288         """
292 -         try:
293 -             fpin.seek(offset - sizeEndCentDir64Locator, 2)
294 -         except OSError:

```

```

295 - # If the seek fails, the file is not large enough to contain a ZIP64
289 + offset -= sizeEndCentDir64Locator
290 + if offset < 0:
291 + # The file is not large enough to contain a ZIP64
296 292 # end-of-archive record, so just return the end record we were given.
297 293 return endrec
298 -
294 + fpin.seek(offset)
299 295 data = fpin.read(sizeEndCentDir64Locator)
300 296 if len(data) != sizeEndCentDir64Locator:
301 - return endrec
297 + raise OSError("Unknown I/O error")
302 298 sig, diskno, reloff, disks = struct.unpack(structEndArchive64Locator,
data)
303 299 if sig != stringEndArchive64Locator:
304 300 return endrec
305 301
306 302 if diskno != 0 or disks > 1:
307 303 raise BadZipFile("zipfiles that span multiple disks are not
supported")
308 304
309 - # Assume no 'zip64 extensible data'
310 - fpin.seek(offset - sizeEndCentDir64Locator - sizeEndCentDir64, 2)
305 + offset -= sizeEndCentDir64
306 + if reloff > offset:
307 + raise BadZipFile("Corrupt zip64 end of central directory locator")
308 + # First, check the assumption that there is no prepended data.
309 + fpin.seek(reloff)
310 + extrasz = offset - reloff
311 311 data = fpin.read(sizeEndCentDir64)
312 312 if len(data) != sizeEndCentDir64:
313 - return endrec
313 + raise OSError("Unknown I/O error")
314 + if not data.startswith(stringEndArchive64) and reloff != offset:
315 + # Since we already have seen the Zip64 EOCD Locator, it's
316 + # possible we got here because there is prepended data.
317 + # Assume no 'zip64 extensible data'
318 + fpin.seek(offset)
319 + extrasz = 0
320 + data = fpin.read(sizeEndCentDir64)

```

321	+	<code>if len(data) != sizeEndCentDir64:</code>
322	+	<code>raise OSError("Unknown I/O error")</code>
323	+	<code>if not data.startswith(stringEndArchive64):</code>
324	+	<code>raise BadZipFile("Zip64 end of central directory record not found")</code>
325	+	
314	326	<code>sig, sz, create_version, read_version, disk_num, disk_dir, \</code>
315	327	<code>dircount, dircount2, dirsiz, diroffset = \</code>
316	328	<code>struct.unpack(structEndArchive64, data)</code>
317	-	<code>if sig != stringEndArchive64:</code>
318	-	<code>return endrec</code>
329	+	<code>if (diroffset + dirsiz != reloff or</code>
330	+	<code>sz + 12 != sizeEndCentDir64 + extrasz):</code>
331	+	<code>raise BadZipFile("Corrupt zip64 end of central directory record")</code>
319	332	
320	333	<code># Update the original endrec using data from the ZIP64 record</code>
321	334	<code>endrec[_ECD_SIGNATURE] = sig</code>
↕		<code>@@ -325,6 +338,7 @@ def _EndRecData64(fpin, offset, endrec):</code>
325	338	<code>endrec[_ECD_ENTRIES_TOTAL] = dircount2</code>
326	339	<code>endrec[_ECD_SIZE] = dirsiz</code>
327	340	<code>endrec[_ECD_OFFSET] = diroffset</code>
341	+	<code>endrec[_ECD_LOCATION] = offset - extrasz</code>
328	342	<code>return endrec</code>
329	343	
330	344	
↓		<code>@@ -358,7 +372,7 @@ def _EndRecData(fpin):</code>
358	372	<code>endrec.append(filesiz - sizeEndCentDir)</code>
359	373	
360	374	<code># Try to read the "Zip64 end of central directory" structure</code>
361	-	<code>return _EndRecData64(fpin, -sizeEndCentDir, endrec)</code>
375	+	<code>return _EndRecData64(fpin, filesiz - sizeEndCentDir, endrec)</code>
362	376	
363	377	<code># Either this is not a ZIP file, or it is a ZIP file with an archive</code>
364	378	<code># comment. Search the end of the file for the "end of central directory"</code>
↕		<code>@@ -382,8 +396,7 @@ def _EndRecData(fpin):</code>
382	396	<code>endrec.append(maxCommentStart + start)</code>
383	397	
384	398	<code># Try to read the "Zip64 end of central directory" structure</code>
385	-	<code>return _EndRecData64(fpin, maxCommentStart + start - filesiz,</code>
386	-	<code>endrec)</code>

```

399 +         return _EndRecData64(fpin, maxCommentStart + start, endrec)
387 400
388 401         # Unable to find a valid end of central directory structure
389 402         return None
    ↓
    ↑
@@ -2142,7 +2155,7 @@ def _write_end_record(self):
                " would require ZIP64 extensions")
2142 2155
2143 2156         zip64endrec = struct.pack(
2144 2157             structEndArchive64, stringEndArchive64,
2145 -         44, 45, 45, 0, 0, centDirCount, centDirCount,
+         sizeEndCentDir64 - 12, 45, 45, 0, 0, centDirCount,
2158 +         centDirCount,
2146 2159             centDirSize, centDirOffset)
2147 2160         self.fp.write(zip64endrec)
2148 2161
    ↓

```

...5-10-07-19-31-34.gh-issue-139700.vNHU10.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