

python / cpython Public

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

Commit 1d29afb



miss-islington and serhiy-storchaka authored on Oct 8, 2025 · ✖ 58 / 70 · Verified

[3.11] [gh-139700](#): Check consistency of the zip64 end of central directory record
 ([GH-139702](#)) ([GH-139708](#)) ([GH-139713](#))

(cherry picked from commit [333d4a6](#))

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>

[3.11](#) (#98846, #139713) · v3.11.15 v3.11.14

1 parent [22d5724](#) commit 1d29afb

3 files changed +113 -23 ●●●●●

[↑ Top](#)

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

```

  ▾ Lib/test/test_zipfile.py
  ↑ @ -887,6 +887,8 @@ def make_zip64_file(
  
```

```

887 887         self, file_size_64_set=False, file_size_extra=False,
888 888         compress_size_64_set=False, compress_size_extra=False,
889 889         header_offset_64_set=False, header_offset_extra=False,
890 890         +         extensible_data=b'',
891 891         +         end_of_central_dir_size=None, offset_to_end_of_central_dir=None,
890 892     ):
891 893         """Generate bytes sequence for a zip with (incomplete) zip64 data.
892 894
@@ -940,6 +942,12 @@ def make_zip64_file(
940 942
941 943         central_dir_size = struct.pack('<Q', 58 + 8 *
len(central_zip64_fields))
942 944         offset_to_central_dir = struct.pack('<Q', 50 + 8 *
len(local_zip64_fields))
945 945         +         if end_of_central_dir_size is None:
946 946         +             end_of_central_dir_size = 44 + len(extensible_data)
947 947         +         if offset_to_end_of_central_dir is None:
948 948         +             offset_to_end_of_central_dir = (108
949 949         +                 + 8 * len(local_zip64_fields)
950 950         +                 + 8 * len(central_zip64_fields))
943 951
944 952         local_extra_length = struct.pack("<H", 4 + 8 *
len(local_zip64_fields))
945 953         central_extra_length = struct.pack("<H", 4 + 8 *
len(central_zip64_fields))
@@ -968,14 +976,17 @@ def make_zip64_file(
968 976         + filename
969 977         + central_extra
970 978         # Zip64 end of central directory
971 978         -         + b"PK\x06\x06,\x00\x00\x00\x00\x00\x00\x00-\x00-"
972 978         -         + b"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
979 978         +         + b"PK\x06\x06"
980 978         +         + struct.pack('<Q', end_of_central_dir_size)
981 978         +         + b"-\x00-
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00"
973 982         + b"\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00"
974 983         + central_dir_size
975 984         + offset_to_central_dir

```

985	+	+ extensible_data
976	986	# Zip64 end of central directory locator
977	-	+ b"PK\x06\x07\x00\x00\x00\x001\x00\x00\x00\x00\x00\x01"
978	-	+ b"\x00\x00\x00"
987	+	+ b"PK\x06\x07\x00\x00\x00\x00"
988	+	+ struct.pack('<Q', offset_to_end_of_central_dir)
989	+	+ b"\x01\x00\x00\x00"
979	990	# end of central directory
980	991	+ b"PK\x05\x06\x00\x00\x00\x00\x01\x00\x01\x00:\x00\x00\x002\x00"
981	992	+ b"\x00\x00\x00\x00"
⋮ ↓ ↑ ⋮		@@ -1006,6 +1017,7 @@ def test_bad_zip64_extra(self):
1006	1017	with self.assertRaises(zipfile.BadZipFile) as e:
1007	1018	zipfile.ZipFile(io.BytesIO(missing_file_size_extra))
1008	1019	self.assertIn('file size', str(e.exception).lower())
1020	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_file_size_extra)))
1009	1021	
1010	1022	# zip64 file size present, zip64 compress size present, one field in
1011	1023	# extra, expecting two, equals missing compress size.
⋮ ↕ ⋮		@@ -1017,6 +1029,7 @@ def test_bad_zip64_extra(self):
1017	1029	with self.assertRaises(zipfile.BadZipFile) as e:
1018	1030	zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1019	1031	self.assertIn('compress size', str(e.exception).lower())
1032	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1020	1033	
1021	1034	# zip64 compress size present, no fields in extra, expecting one,
1022	1035	# equals missing compress size.
⋮ ↕ ⋮		@@ -1026,6 +1039,7 @@ def test_bad_zip64_extra(self):
1026	1039	with self.assertRaises(zipfile.BadZipFile) as e:
1027	1040	zipfile.ZipFile(io.BytesIO(missing_compress_size_extra))
1028	1041	self.assertIn('compress size', str(e.exception).lower())
1042	+	self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_compress_size_extra)))
1029	1043	
1030	1044	# zip64 file size present, zip64 compress size present, zip64 header
1031	1045	# offset present, two fields in extra, expecting three, equals
		missing
⋮ ↕ ⋮		@@ -1040,6 +1054,7 @@ def test_bad_zip64_extra(self):

```

1040 1054         with self.assertRaises(zipfile.BadZipFile) as e:
1041 1055             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1042 1056         self.assertIn('header offset', str(e.exception).lower())
1057 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1043 1058
1044 1059         # zip64 compress size present, zip64 header offset present, one field
1045 1060         # in extra, expecting two, equals missing header offset
@@ -1052,6 +1067,7 @@ def test_bad_zip64_extra(self):
1052 1067         with self.assertRaises(zipfile.BadZipFile) as e:
1053 1068             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1054 1069         self.assertIn('header offset', str(e.exception).lower())
1070 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1055 1071
1056 1072         # zip64 file size present, zip64 header offset present, one field in
1057 1073         # extra, expecting two, equals missing header offset
@@ -1064,6 +1080,7 @@ def test_bad_zip64_extra(self):
1064 1080         with self.assertRaises(zipfile.BadZipFile) as e:
1065 1081             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1066 1082         self.assertIn('header offset', str(e.exception).lower())
1083 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1067 1084
1068 1085         # zip64 header offset present, no fields in extra, expecting one,
1069 1086         # equals missing header offset
@@ -1075,6 +1092,63 @@ def test_bad_zip64_extra(self):
1075 1092         with self.assertRaises(zipfile.BadZipFile) as e:
1076 1093             zipfile.ZipFile(io.BytesIO(missing_header_offset_extra))
1077 1094         self.assertIn('header offset', str(e.exception).lower())
1095 +
self.assertTrue(zipfile.is_zipfile(io.BytesIO(missing_header_offset_extra)))
1096 +
1097 +         def test_bad_zip64_end_of_central_dir(self):
1098 +             zipdata = self.make_zip64_file(end_of_central_dir_size=0)
1099 +             with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1100 +                 zipfile.ZipFile(io.BytesIO(zipdata))
1101 +             self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1102 +
1103 +             zipdata = self.make_zip64_file(end_of_central_dir_size=100)

```

```
1104 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1105 +             zipfile.ZipFile(io.BytesIO(zipdata))
1106 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1107 +
1108 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=0)
1109 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*record'):
1110 +             zipfile.ZipFile(io.BytesIO(zipdata))
1111 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1112 +
1113 +         zipdata = self.make_zip64_file(offset_to_end_of_central_dir=1000)
1114 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'Corrupt.*locator'):
1115 +             zipfile.ZipFile(io.BytesIO(zipdata))
1116 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1117 +
1118 +     def test_zip64_end_of_central_dir_record_not_found(self):
1119 +         zipdata = self.make_zip64_file()
1120 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1121 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1122 +             zipfile.ZipFile(io.BytesIO(zipdata))
1123 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1124 +
1125 +         zipdata = self.make_zip64_file(
1126 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1127 +         zipdata = zipdata.replace(b"PK\x06\x06", b'\x00'*4)
1128 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1129 +             zipfile.ZipFile(io.BytesIO(zipdata))
1130 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(zipdata)))
1131 +
1132 +     def test_zip64_extensible_data(self):
1133 +         # These values are what is set in the make_zip64_file method.
1134 +         expected_file_size = 8
1135 +         expected_compress_size = 8
1136 +         expected_header_offset = 0
1137 +         expected_content = b"test1234"
1138 +
1139 +         zipdata = self.make_zip64_file(
1140 +             extensible_data=b'\xca\xfe\x04\x00\x00\x00data')
1141 +         with zipfile.ZipFile(io.BytesIO(zipdata)) as zf:
1142 +             zinfo = zf.infolist()[0]
1143 +             self.assertEqual(zinfo.file_size, expected_file_size)
```

```

1144 +         self.assertEqual(zinfo.compress_size, expected_compress_size)
1145 +         self.assertEqual(zinfo.header_offset, expected_header_offset)
1146 +         self.assertEqual(zf.read(zinfo), expected_content)
1147 +         self.assertTrue(zipfile.is_zipfile(io.BytesIO(zipdata)))
1148 +
1149 +         with self.assertRaisesRegex(zipfile.BadZipFile, 'record not found'):
1150 +             zipfile.ZipFile(io.BytesIO(b'prepended' + zipdata))
1151 +         self.assertFalse(zipfile.is_zipfile(io.BytesIO(b'prepended' +
zipdata)))
1078 1152
1079 1153         def test_generated_valid_zip64_extra(self):
1080 1154             # These values are what is set in the make_zip64_file method.

```

Lib/zipfile.py

```

@@ -236,41 +236,57 @@ def is_zipfile(filename):
236 236         else:
237 237             with open(filename, "rb") as fp:
238 238                 result = _check_zipfile(fp)
239 -         except OSError:
239 +         except (OSError, BadZipFile):
240 240             pass
241 241             return result
242 242
243 243         def _EndRecData64(fpin, offset, endrec):
244 244             """
245 245             Read the ZIP64 end-of-archive records and use that to update endrec
246 246             """
247 -         try:
248 -             fpin.seek(offset - sizeEndCentDir64Locator, 2)
249 -         except OSError:
250 -             # If the seek fails, the file is not large enough to contain a ZIP64
247 +         offset -= sizeEndCentDir64Locator
248 +         if offset < 0:
249 +             # The file is not large enough to contain a ZIP64
251 250             # end-of-archive record, so just return the end record we were given.
252 251             return endrec
253 -
252 +         fpin.seek(offset)
254 253         data = fpin.read(sizeEndCentDir64Locator)

```

```

255 254         if len(data) != sizeEndCentDir64Locator:
256 -         return endrec
255 +         raise OSError("Unknown I/O error")
257 256         sig, diskno, reloff, disks = struct.unpack(structEndArchive64Locator,
data)
258 257         if sig != stringEndArchive64Locator:
259 258             return endrec
260 259
261 260         if diskno != 0 or disks > 1:
262 261             raise BadZipFile("zipfiles that span multiple disks are not
supported")
263 262
264 -         # Assume no 'zip64 extensible data'
265 -         fpin.seek(offset - sizeEndCentDir64Locator - sizeEndCentDir64, 2)
263 +         offset -= sizeEndCentDir64
264 +         if reloff > offset:
265 +             raise BadZipFile("Corrupt zip64 end of central directory locator")
266 +         # First, check the assumption that there is no prepended data.
267 +         fpin.seek(reloff)
268 +         extrasz = offset - reloff
266 269         data = fpin.read(sizeEndCentDir64)
267 270         if len(data) != sizeEndCentDir64:
268 -         return endrec
271 +         raise OSError("Unknown I/O error")
272 +         if not data.startswith(stringEndArchive64) and reloff != offset:
273 +             # Since we already have seen the Zip64 EOCD Locator, it's
274 +             # possible we got here because there is prepended data.
275 +             # Assume no 'zip64 extensible data'
276 +             fpin.seek(offset)
277 +             extrasz = 0
278 +             data = fpin.read(sizeEndCentDir64)
279 +             if len(data) != sizeEndCentDir64:
280 +                 raise OSError("Unknown I/O error")
281 +             if not data.startswith(stringEndArchive64):
282 +                 raise BadZipFile("Zip64 end of central directory record not found")
283 +
269 284         sig, sz, create_version, read_version, disk_num, disk_dir, \
270 285         dircount, dircount2, dirsized, diroffset = \
271 286         struct.unpack(structEndArchive64, data)
272 -         if sig != stringEndArchive64:

```

273	-	<code>return endrec</code>
287	+	<code>if (diroffset + dirsize != reloff or</code>
288	+	<code>sz + 12 != sizeEndCentDir64 + extrasz):</code>
289	+	<code>raise BadZipFile("Corrupt zip64 end of central directory record")</code>
274	290	
275	291	<code># Update the original endrec using data from the ZIP64 record</code>
276	292	<code>endrec[_ECD_SIGNATURE] = sig</code>
↕		<code>@@ -280,6 +296,7 @@ def _EndRecData64(fpin, offset, endrec):</code>
280	296	<code>endrec[_ECD_ENTRIES_TOTAL] = dircount2</code>
281	297	<code>endrec[_ECD_SIZE] = dirsize</code>
282	298	<code>endrec[_ECD_OFFSET] = diroffset</code>
299	+	<code>endrec[_ECD_LOCATION] = offset - extrasz</code>
283	300	<code>return endrec</code>
284	301	
285	302	
↓		<code>@@ -313,7 +330,7 @@ def _EndRecData(fpin):</code>
↑		
313	330	<code>endrec.append(filesize - sizeEndCentDir)</code>
314	331	
315	332	<code># Try to read the "Zip64 end of central directory" structure</code>
316	-	<code>return _EndRecData64(fpin, -sizeEndCentDir, endrec)</code>
333	+	<code>return _EndRecData64(fpin, filesize - sizeEndCentDir, endrec)</code>
317	334	
318	335	<code># Either this is not a ZIP file, or it is a ZIP file with an archive</code>
319	336	<code># comment. Search the end of the file for the "end of central directory"</code>
↕		<code>@@ -337,8 +354,7 @@ def _EndRecData(fpin):</code>
337	354	<code>endrec.append(maxCommentStart + start)</code>
338	355	
339	356	<code># Try to read the "Zip64 end of central directory" structure</code>
340	-	<code>return _EndRecData64(fpin, maxCommentStart + start - filesize,</code>
341	-	<code>endrec)</code>
357	+	<code>return _EndRecData64(fpin, maxCommentStart + start, endrec)</code>
342	358	
343	359	<code># Unable to find a valid end of central directory structure</code>
344	360	<code>return None</code>
↓		<code>@@ -1386,9 +1402,6 @@ def _RealGetContents(self):</code>
↑		
1386	1402	
1387	1403	<code># "concat" is zero, unless zip was concatenated to another file</code>
1388	1404	<code>concat = endrec[_ECD_LOCATION] - size_cd - offset_cd</code>

