

python / cpython Public

<> Code Issues 5k+ Pull requests 2.1k Actions Projects Security and q

Commit 69b4387



miss-islington and serhiy-storchaka authored on May 13, 2025 · ✖ 50 / 68 · Verified



[3.14] [gh-133767](#): Fix use-after-free in the unicode-escape decoder with an error handler ([GH-129648](#)) ([GH-133942](#))

If the error handler is used, a new bytes object is created to set as the object attribute of UnicodeDecodeError, and that bytes object then replaces the original data. A pointer to the decoded data will become invalid after destroying that temporary bytes object. So we need other way to return the first invalid escape from `_PyUnicode_DecodeUnicodeEscapeInternal()`.

`_PyBytes_DecodeEscape()` does not have such issue, because it does not use the error handlers registry, but it should be changed for compatibility with `_PyUnicode_DecodeUnicodeEscapeInternal()`.
(cherry picked from commit [9f69a58](#))

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

[3.14](#) (#133942) · v3.14.4 ... v3.14.0b2

1 parent [f0a7a6c](#) commit 69b4387

9 files changed

+160 -80 ■■■■■■

Top

- ▾ 📁 Doc/data
 - python3.14.abi
- ▾ 📁 Include/internal
 - pycore_bytesobject.h
 - pycore_unicodeobject.h
- ▾ 📁 Lib/test
 - test_codeccallbacks.py

- test_codec.py
- Misc/NEWS.d/next/Security
 - 2025-05-09-20-22-54.gh-issue-133767.kN2i3Q.rst
- Objects
 - bytesobject.c
 - unicodeobject.c
- Parser
 - string_parser.c



Search within code



Doc/data/python3.14.abi



↑		@@ -1130,7 +1130,6 @@
1130	1130	<elf-symbol name='_PyBytesWriter_Prepares' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1131	1131	<elf-symbol name='_PyBytesWriter_Resize' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1132	1132	<elf-symbol name='_PyBytesWriter_WriteBytes' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1133	-	<elf-symbol name='_PyBytes_DecodeEscape' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1134	1133	<elf-symbol name='_PyBytes_Find' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1135	1134	<elf-symbol name='_PyBytes_FromData' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1136	1135	<elf-symbol name='_PyBytes_FromXIData' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
↓		@@ -1448,7 +1447,6 @@
1448	1447	<elf-symbol name='_PyUnicode_AsUTF8String' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />

1449	1448	<elf-symbol name='_PyUnicode_CheckConsistency' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1450	1449	<elf-symbol name='_PyUnicode_Copy' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1451	-	<elf-symbol name='_PyUnicode_DecodeUnicodeEscapeInternal' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1452	1450	<elf-symbol name='_PyUnicode_EncodeUTF16' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1453	1451	<elf-symbol name='_PyUnicode_EncodeUTF32' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
1454	1452	<elf-symbol name='_PyUnicode_Equal' type='func-type' binding='global-binding' visibility='default-visibility' is-defined='yes' />
		@@ -24180,21 +24178,6 @@
24180	24178	<parameter type-id='type-id-6' />
24181	24179	<return type-id='type-id-5' />
24182	24180	</function-decl>
24183	-	<function-decl name='_PyBytes_DecodeEscape' mangled-name='_PyBytes_DecodeEscape' filepath='./Include/internal/pycore_bytesobject.h' line='23' column='1' visibility='default' binding='global' size-in-bits='64' elf-symbol-id='_PyBytes_DecodeEscape'>
24184	-	<parameter type-id='type-id-4' />
24185	-	<parameter type-id='type-id-7' />
24186	-	<parameter type-id='type-id-4' />
24187	-	<parameter type-id='type-id-266' />
24188	-	<return type-id='type-id-6' />
24189	-	</function-decl>
24190	-	<function-decl name='_PyUnicode_DecodeUnicodeEscapeInternal' mangled-name='_PyUnicode_DecodeUnicodeEscapeInternal' filepath='./Include/internal/pycore_unicodeobject.h' line='142' column='1' visibility='default' binding='global' size-in-bits='64' elf-symbol-id='_PyUnicode_DecodeUnicodeEscapeInternal'>
24191	-	<parameter type-id='type-id-4' />
24192	-	<parameter type-id='type-id-7' />
24193	-	<parameter type-id='type-id-4' />

```

24194 - <parameter type-id='type-id-8' />
24195 - <parameter type-id='type-id-266' />
24196 - <return type-id='type-id-6' />
24197 - </function-decl>
24198 24181 <function-decl name='_PyErr_BadInternalCall' mangled-
name='_PyErr_BadInternalCall' filepath='./Include/pyerrors.h' line='223'
column='1' visibility='default' binding='global' size-in-bits='64' elf-
symbol-id='_PyErr_BadInternalCall'>
24199 24182 <parameter type-id='type-id-4' />
24200 24183 <parameter type-id='type-id-5' />

```



Include/internal/pycore_bytesobject.h



```

↑ ... @@ -20,8 +20,9 @@ extern PyObject* _PyBytes_FromHex(
20 20
21 21 // Helper for PyBytes_DecodeEscape that detects invalid escape chars.
22 22 // Export for test_peg_generator.
23 - PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape(const char *, Py_ssize_t,
24 -                                             const char *, const char **);
23 + PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape2(const char *, Py_ssize_t,
24 +                                             const char *,
25 +                                             int *, const char **);
25 26
26 27
27 28 // Substring Search.

```



Include/internal/pycore_unicodeobject.h



```

↑ ... @@ -139,14 +139,18 @@ extern PyObject*
_PyUnicode_DecodeUnicodeEscapeStateful(
139 139 // Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
140 140 // chars.
141 141 // Export for test_peg_generator.
142 - PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
142 + PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
143 143     const char *string, /* Unicode-Escape encoded string */
144 144     Py_ssize_t length, /* size of string */
145 145     const char *errors, /* error handling */
146 146     Py_ssize_t *consumed, /* bytes consumed */

```

```

147 -     const char **first_invalid_escape); /* on return, points to first
148 -         invalid escaped char in
149 -         string. */
147 +     int *first_invalid_escape_char, /* on return, if not -1, contain the first
148 +         invalid escaped char (<= 0xff) or
149 +         invalid
150 +         octal escape (> 0xff) in string. */
151 +     const char **first_invalid_escape_ptr); /* on return, if not NULL, may
152 +         point to the first invalid escaped
153 +         char in string.
154 +         May be NULL if errors is not NULL. */
150 154
151 155     /* --- Raw-Unicode-Escape Codecs -----
152 156     */

```



Lib/test/test_codeccallbacks.py



```
@@ -2,6 +2,7 @@
```

```

2 2     import codecs
3 3     import html.entities
4 4     import itertools
5 5 +    import re
5 6     import sys
6 7     import unicodedata
7 8     import unittest

```



```
@@ -1125,7 +1126,7 @@ def test_bug828737(self):
```



```

1125 1126         text = 'abc<def>ghi'*n
1126 1127         text.translate(charmap)
1127 1128

```

```

1128 -     def test_mutatingdecodehandler(self):
1129 +     def test_mutating_decode_handler(self):
1129 1130         baddata = [
1130 1131             ("ascii", b"\xff"),
1131 1132             ("utf-7", b"++"),

```



```
@@ -1160,6 +1161,42 @@ def mutating(exc):
```



```

1160 1161         for (encoding, data) in baddata:

```

```
1161 1162         self.assertEqual(data.decode(encoding, "test.mutating"),
1163                          "\u4242")
1164 1163
1164 +     def test_mutating_decode_handler_unicode_escape(self):
1165 +         decode = codecs.unicode_escape_decode
1166 +         def mutating(exc):
1167 +             if isinstance(exc, UnicodeDecodeError):
1168 +                 r = data.get(exc.object[:exc.end])
1169 +                 if r is not None:
1170 +                     exc.object = r[0] + exc.object[exc.end:]
1171 +                     return ('\u0404', r[1])
1172 +                 raise AssertionError("don't know how to handle %r" % exc)
1173 +
1174 +         codecs.register_error('test.mutating2', mutating)
1175 +         data = {
1176 +             br'\x0': (b'\\', 0),
1177 +             br'\x3': (b'xxx\\', 3),
1178 +             br'\x5': (b'x\\', 1),
1179 +         }
1180 +         def check(input, expected, msg):
1181 +             with self.assertWarns(DeprecationWarning) as cm:
1182 +                 self.assertEqual(decode(input, 'test.mutating2'), (expected,
1183 +                             len(input)))
1184 +                 self.assertIn(msg, str(cm.warning))
1185 +             check(br'\x0n\z', '\u0404\n\\z', r'"z" is an invalid escape
1186 +                 sequence')
1187 +             check(br'\x0n\501', '\u0404\n\u0141', r'"501" is an invalid octal
1188 +                 escape sequence')
1189 +             check(br'\x0z', '\u0404\\z', r'"z" is an invalid escape sequence')
1190 +             check(br'\x3n\zr', '\u0404\n\\zr', r'"z" is an invalid escape
1191 +                 sequence')
1192 +             check(br'\x3zr', '\u0404\\zr', r'"z" is an invalid escape sequence')
1193 +             check(br'\x3z5', '\u0404\\z5', r'"z" is an invalid escape sequence')
1194 +             check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r'"z" is an invalid
1195 +                 escape sequence')
1196 +             check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r'"z" is an
1197 +                 invalid escape sequence')
```

```

1195 +         check(br'\x5n\z', '\u0404\n\\z', r'"z" is an invalid escape
      +         sequence')
1196 +         check(br'\x5n\501', '\u0404\n\u0141', r'"501" is an invalid octal
      +         escape sequence')
1197 +         check(br'\x5z', '\u0404\\z', r'"z" is an invalid escape sequence')
1198 +         check(memoryview(br'\x5zy')[:-1], '\u0404\\z', r'"z" is an invalid
      +         escape sequence')
1199 +
1163 1200         # issue32583
1164 1201         def test_crashing_decode_handler(self):
1165 1202             # better generating one more character to fill the extra space slot
      ↓

```

```

Lib/test/test_codec.py
  ↑
@@ -1196,23 +1196,39 @@ def test_escape(self):
1196 1196         check(br"[\1010]", b"[A0]")
1197 1197         check(br"[\x41]", b"[A]")
1198 1198         check(br"[\x410]", b"[A0]")
1199 +
1200 +         def test_warnings(self):
1201 +             decode = codecs.escape_decode
1202 +             check = coding_checker(self, decode)
1199 1203         for i in range(97, 123):
1200 1204             b = bytes([i])
1201 1205             if b not in b'abfnrtvx':
1202 -                 with self.assertWarns(DeprecationWarning):
1206 +                 with self.assertWarnsRegex(DeprecationWarning,
1207 +                 r'"\\%c" is an invalid escape sequence' % i):
1203 1208                     check(b"\" + b, b"\" + b)
1204 -                 with self.assertWarns(DeprecationWarning):
1209 +                 with self.assertWarnsRegex(DeprecationWarning,
1210 +                 r'"\\%c" is an invalid escape sequence' % (i-32)):
1205 1211                     check(b"\" + b.upper(), b"\" + b.upper())
1206 -                 with self.assertWarns(DeprecationWarning):
1212 +                 with self.assertWarnsRegex(DeprecationWarning,
1213 +                 r'"\\8" is an invalid escape sequence'):
1207 1214                     check(br"\8", b"\8")
1208 1215                 with self.assertWarns(DeprecationWarning):
1209 1216                     check(br"\9", b"\9")
1210 -                 with self.assertWarns(DeprecationWarning):

```

1217	+	<code>with self.assertWarnsRegex(DeprecationWarning,</code>
1218	+	<code> r'"\\xfa" is an invalid escape sequence') as cm:</code>
1211	1219	<code> check(b"\\xfa", b"\\xfa")</code>
1212	1220	<code> for i in range(0x400, 0x1000):</code>
1213	-	<code> with self.assertWarns(DeprecationWarning):</code>
1221	+	<code> with self.assertWarnsRegex(DeprecationWarning,</code>
1222	+	<code> r'"\\%o" is an invalid octal escape sequence' % i):</code>
1214	1223	<code> check(rb'\\%o' % i, bytes([i & 0x377]))</code>
1215	1224	
1225	+	<code> with self.assertWarnsRegex(DeprecationWarning,</code>
1226	+	<code> r'"\\z" is an invalid escape sequence'):</code>
1227	+	<code> self.assertEqual(decode(br'\\z', 'ignore'), (b'\\z', 4))</code>
1228	+	<code> with self.assertWarnsRegex(DeprecationWarning,</code>
1229	+	<code> r'"\\501" is an invalid octal escape sequence'):</code>
1230	+	<code> self.assertEqual(decode(br'\\x\\501', 'ignore'), (b'A', 6))</code>
1231	+	
1216	1232	<code>def test_errors(self):</code>
1217	1233	<code> decode = codecs.escape_decode</code>
1218	1234	<code> self.assertRaises(ValueError, decode, br"\\x")</code>
		<code>@@ -2661,24 +2677,40 @@ def test_escape_decode(self):</code>
2661	2677	<code> check(br"\\x410", "[A0]")</code>
2662	2678	<code> check(br"\\u20ac", "\\u20ac")</code>
2663	2679	<code> check(br"\\U0001d120", "\\U0001d120")</code>
2680	+	
2681	+	<code>def test_decode_warnings(self):</code>
2682	+	<code> decode = codecs.unicode_escape_decode</code>
2683	+	<code> check = coding_checker(self, decode)</code>
2664	2684	<code> for i in range(97, 123):</code>
2665	2685	<code> b = bytes([i])</code>
2666	2686	<code> if b not in b'abfnrtuvx':</code>
2667	-	<code> with self.assertWarns(DeprecationWarning):</code>
2687	+	<code> with self.assertWarnsRegex(DeprecationWarning,</code>
2688	+	<code> r'"\\%c" is an invalid escape sequence' % i):</code>
2668	2689	<code> check(b"\\\" + b, "\\\" + chr(i))</code>
2669	2690	<code> if b.upper() not in b'UN':</code>
2670	-	<code> with self.assertWarns(DeprecationWarning):</code>
2691	+	<code> with self.assertWarnsRegex(DeprecationWarning,</code>
2692	+	<code> r'"\\%c" is an invalid escape sequence' % (i-32)):</code>
2671	2693	<code> check(b"\\\" + b.upper(), "\\\" + chr(i-32))</code>

```

2672 -         with self.assertWarns(DeprecationWarning):
2694 +         with self.assertWarnsRegex(DeprecationWarning,
2695 +             r'"\\8" is an invalid escape sequence'):
2673 2696             check(br"\8", "\\8")
2674 2697             with self.assertWarns(DeprecationWarning):
2675 2698                 check(br"\9", "\\9")
2676 -         with self.assertWarns(DeprecationWarning):
2699 +         with self.assertWarnsRegex(DeprecationWarning,
2700 +             r'"\\xfa" is an invalid escape sequence') as cm:
2677 2701             check(b"\\xfa", "\\xfa")
2678 2702             for i in range(0o400, 0o1000):
2679 -         with self.assertWarns(DeprecationWarning):
2703 +         with self.assertWarnsRegex(DeprecationWarning,
2704 +             r'"\\%o" is an invalid octal escape sequence' % i):
2680 2705             check(rb'\%o' % i, chr(i))
2681 2706
2707 +         with self.assertWarnsRegex(DeprecationWarning,
2708 +             r'"\\z" is an invalid escape sequence'):
2709 +             self.assertEqual(decode(br'\x\z', 'ignore'), ('\\z', 4))
2710 +         with self.assertWarnsRegex(DeprecationWarning,
2711 +             r'"\\501" is an invalid octal escape sequence'):
2712 +             self.assertEqual(decode(br'\x\501', 'ignore'), ('\u0141', 6))
2713 +
2682 2714         def test_decode_errors(self):
2683 2715             decode = codecs.unicode_escape_decode
2684 2716             for c, d in (b'x', 2), (b'u', 4), (b'U', 4):

```



...5-05-09-20-22-54.gh-issue-133767.kN2i3Q.rst



... @@ -0,0 +1,2 @@

1 + Fix use-after-free in the "unicode-escape" decoder with a non-"strict" error

2 + handler.

Objects/bytesobject.c



```

@@ -1075,10 +1075,11 @@ _PyBytes_FormatEx(const char *format, Py_ssize_t
format_len,

```

1075 1075 }

1076 1076

1077 1077 /* Unescape a backslash-escaped string. */

1078	-	PyObject *_PyBytes_DecodeEscape(const char *s,
1078	+	PyObject *_PyBytes_DecodeEscape2(const char *s,
1079	1079	Py_ssize_t len,
1080	1080	const char *errors,
1081	-	const char **first_invalid_escape)
1081	+	int *first_invalid_escape_char,
1082	+	const char **first_invalid_escape_ptr)
1082	1083	{
1083	1084	int c;
1084	1085	char *p;
		@@ -1092,7 +1093,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1092	1093	return NULL;
1093	1094	writer.overallocate = 1;
1094	1095	
1095	-	*first_invalid_escape = NULL;
1096	+	*first_invalid_escape_char = -1;
1097	+	*first_invalid_escape_ptr = NULL;
1096	1098	
1097	1099	end = s + len;
1098	1100	while (s < end) {
		@@ -1130,9 +1132,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1130	1132	c = (c<<3) + *s++ - '0';
1131	1133	}
1132	1134	if (c > 0377) {
1133	-	if (*first_invalid_escape == NULL) {
1134	-	*first_invalid_escape = s-3; /* Back up 3 chars, since
		we've
1135	-	already incremented s. */
1135	+	if (*first_invalid_escape_char == -1) {
1136	+	*first_invalid_escape_char = c;
1137	+	/* Back up 3 chars, since we've already incremented s. */
1138	+	*first_invalid_escape_ptr = s - 3;
1136	1139	}
1137	1140	}
1138	1141	*p++ = c;
		@@ -1173,9 +1176,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1173	1176	break;
1174	1177	

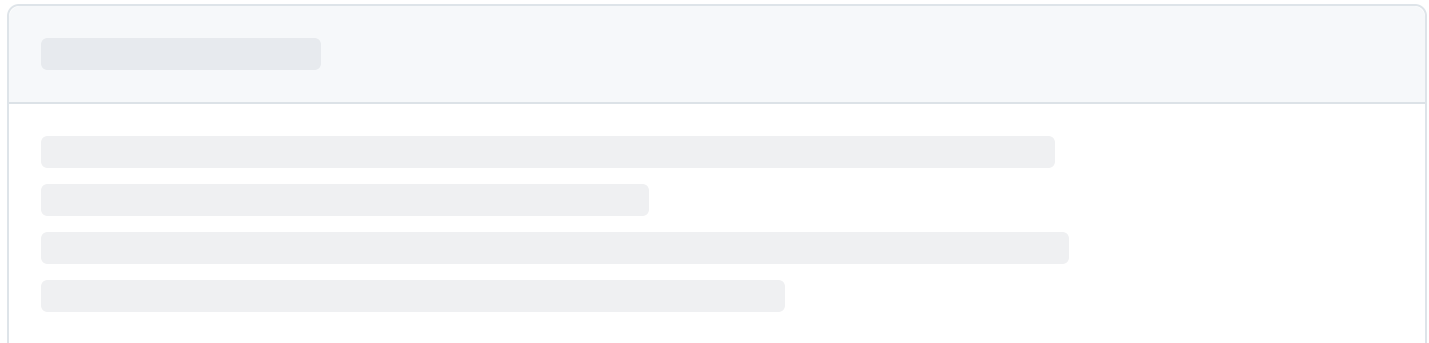
```

1175 1178         default:
1176 -         if (*first_invalid_escape == NULL) {
1177 -             *first_invalid_escape = s-1; /* Back up one char, since we've
1178 -                 already incremented s. */
1179 +         if (*first_invalid_escape_char == -1) {
1180 +             *first_invalid_escape_char = (unsigned char)s[-1];
1181 +             /* Back up one char, since we've already incremented s. */
1182 +             *first_invalid_escape_ptr = s - 1;
1179 1183     }
1180 1184     *p++ = '\\';
1181 1185     s--;
@@ -1195,18 +1199,19 @@ PyObject *PyBytes_DecodeEscape(const char *s,
1195 1199         Py_ssize_t Py_UNUSED(unicode),
1196 1200         const char *Py_UNUSED(recode_encoding))
1197 1201     {
1198 -     const char* first_invalid_escape;
1199 -     PyObject *result = _PyBytes_DecodeEscape(s, len, errors,
1200 -         &first_invalid_escape);
1202 +     int first_invalid_escape_char;
1203 +     const char *first_invalid_escape_ptr;
1204 +     PyObject *result = _PyBytes_DecodeEscape2(s, len, errors,
1205 +         &first_invalid_escape_char,
1206 +         &first_invalid_escape_ptr);
1201 1207     if (result == NULL)
1202 1208         return NULL;
1203 -     if (first_invalid_escape != NULL) {
1204 -         unsigned char c = *first_invalid_escape;
1205 -         if ('4' <= c && c <= '7') {
1209 +         if (first_invalid_escape_char != -1) {
1210 +             if (first_invalid_escape_char > 0xff) {
1206 1211         if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
1207 -             "b\\\"\\%.3s\\\" is an invalid octal escape
sequence. "
1212 +             "b\\\"\\%0\\\" is an invalid octal escape
sequence. "
1208 1213         "Such sequences will not work in the future.
",
1209 -         first_invalid_escape) < 0)
1214 +         first_invalid_escape_char) < 0)
1210 1215     {

```

```
1211 1216 Py_DECREF(result);
1212 1217 return NULL;
@@ -1216,7 +1221,7 @@ PyObject *PyBytes_DecodeEscape(const char *s,
1216 1221 if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
1217 1222     "b\"\\%c\" is an invalid escape sequence. "
1218 1223     "Such sequences will not work in the future.
",
1219 - c) < 0)
1224 + first_invalid_escape_char) < 0)
1220 1225 {
1221 1226 Py_DECREF(result);
1222 1227 return NULL;

```



Comments 0