

python / cpython Public

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

Commit 6279eb8



serhiy-storchaka authored on May 20, 2025 · ✖ 92 / 97 · Verified

[3.13] [gh-133767](#): Fix use-after-free in the unicode-escape decoder with an error handler ([GH-129648](#)) ([GH-133944](#))

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>

3.13 (AcreationOS-Linux/python#2, #133944) · v3.13.13 ... v3.13.4

1 parent [0c0fedf](#) commit 6279eb8


8 files changed

+194 -57 ■■■■■

Top




- ▾ 📁 Include/internal
 - pycore_bytesobject.h
 - pycore_unicodeobject.h
- ▾ 📁 Lib/test
 - test_codeccallbacks.py
 - test_codecs.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





▼ Include/internal/pycore_bytesobject.h ...

```

↑... @@ -20,6 +20,10 @@ 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_DecodeEscape2(const char *, Py_ssize_t,
24 +                                             const char *,
25 +                                             int *, const char **);
26 + // Export for binary compatibility.
23 27 PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape(const char *, Py_ssize_t,
24 28                                             const char *, const char **);
25 29

```

▼ Include/internal/pycore_unicodeobject.h ...

```

↑... @@ -142,6 +142,19 @@ extern PyObject*
_PyUnicode_DecodeUnicodeEscapeStateful(
142 142 // Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
143 143 // chars.
144 144 // Export for test_peg_generator.
145 + PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
146 +     const char *string, /* Unicode-Escape encoded string */
147 +     Py_ssize_t length, /* size of string */
148 +     const char *errors, /* error handling */
149 +     Py_ssize_t *consumed, /* bytes consumed */
150 +     int *first_invalid_escape_char, /* on return, if not -1, contain the first
151 +                                     invalid escaped char (<= 0xff) or
                                     invalid

```

```

152 +         octal escape (> 0xff) in string. */
153 +         const char **first_invalid_escape_ptr); /* on return, if not NULL, may
154 +         point to the first invalid escaped
155 +         char in string.
156 +         May be NULL if errors is not NULL. */
157 + // Export for binary compatibility.

```

```

145 158 PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
146 159     const char *string, /* Unicode-Escape encoded string */
147 160     Py_ssize_t length, /* size of string */

```



Lib/test/test_codeccallbacks.py



@@ -1,6 +1,7 @@

```

1 1 import codecs
2 2 import html.entities
3 3 import itertools
4 + import re
4 5 import sys
5 6 import unicodedata
6 7 import unittest

```



@@ -1124,7 +1125,7 @@ def test_bug828737(self):

```

1124 1125         text = 'abc<def>ghi'*n
1125 1126         text.translate(charmap)
1126 1127

```

```

1127 -     def test_mutatingdecodehandler(self):
1128 +     def test_mutating_decode_handler(self):

```

```

1128 1129         baddata = [
1129 1130             ("ascii", b"\xff"),
1130 1131             ("utf-7", b"++"),

```



@@ -1159,6 +1160,42 @@ def mutating(exc):

```

1159 1160         for (encoding, data) in baddata:
1160 1161             self.assertEqual(data.decode(encoding, "test.mutating"),
1161 1162                 "\u4242")

```

```

1163 +     def test_mutating_decode_handler_unicode_escape(self):
1164 +         decode = codecs.unicode_escape_decode
1165 +         def mutating(exc):
1166 +             if isinstance(exc, UnicodeDecodeError):

```

```

1167 +         r = data.get(exc.object[:exc.end])
1168 +         if r is not None:
1169 +             exc.object = r[0] + exc.object[exc.end:]
1170 +             return ('\u0404', r[1])
1171 +             raise AssertionError("don't know how to handle %r" % exc)
1172 +
1173 +     codecs.register_error('test.mutating2', mutating)
1174 +     data = {
1175 +         br'\x0': (b'\\', 0),
1176 +         br'\x3': (b'xxx\\', 3),
1177 +         br'\x5': (b'x\\', 1),
1178 +     }
1179 +     def check(input, expected, msg):
1180 +         with self.assertWarns(DeprecationWarning) as cm:
1181 +             self.assertEqual(decode(input, 'test.mutating2'), (expected,
1182 + len(input)))
1182 +             self.assertIn(msg, str(cm.warning))
1183 +
1184 +         check(br'\x0n\z', '\u0404n\\z', r"invalid escape sequence '\z'")
1185 +         check(br'\x0n\501', '\u0404n\u0141', r"invalid octal escape sequence
1186 + '\501'")
1186 +         check(br'\x0z', '\u0404\\z', r"invalid escape sequence '\z'")
1187 +
1188 +         check(br'\x3n\zr', '\u0404n\\zr', r"invalid escape sequence '\z'")
1189 +         check(br'\x3zr', '\u0404\\zr', r"invalid escape sequence '\z'")
1190 +         check(br'\x3z5', '\u0404\\z5', r"invalid escape sequence '\z'")
1191 +         check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r"invalid escape
1192 + sequence '\z'")
1192 +         check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r"invalid escape
1193 + sequence '\z'")
1193 +
1194 +         check(br'\x5n\z', '\u0404n\\z', r"invalid escape sequence '\z'")
1195 +         check(br'\x5n\501', '\u0404n\u0141', r"invalid octal escape sequence
1196 + '\501'")
1196 +         check(br'\x5z', '\u0404\\z', r"invalid escape sequence '\z'")
1197 +         check(memoryview(br'\x5zy')[:-1], '\u0404\\z', r"invalid escape
1198 + sequence '\z'")

```

```
1162 1199         # issue32583
```

```
1163 1200         def test_crashing_decode_handler(self):
```

```
1164 1201 # better generating one more character to fill the extra space slot
```



Lib/test/test_codecs.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"invalid escape sequence '\\%c'" % i):
```

```
1203 1208                     check(b"\" + b, b"\" + b)
```

```
1204 -                 with self.assertWarns(DeprecationWarning):
```

```
1209 +                 with self.assertWarnsRegex(DeprecationWarning,
```

```
1210 +                     r"invalid escape sequence '\\%c'" % (i-32)):
```

```
1205 1211                     check(b"\" + b.upper(), b"\" + b.upper())
```

```
1206 -                 with self.assertWarns(DeprecationWarning):
```

```
1212 +                 with self.assertWarnsRegex(DeprecationWarning,
```

```
1213 +                     r"invalid escape sequence '\\8'"):
```

```
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 +                 with self.assertWarnsRegex(DeprecationWarning,
```

```
1218 +                     r"invalid escape sequence '\\\xfa'" as cm:
```

```
1211 1219                     check(b"\\\xfa", b"\\\xfa")
```

```
1212 1220                 for i in range(0o400, 0o1000):
```

```
1213 -                 with self.assertWarns(DeprecationWarning):
```

```
1221 +                 with self.assertWarnsRegex(DeprecationWarning,
```

```
1222 +                     r"invalid octal escape sequence '\\%o'" % i):
```

```
1214 1223                     check(rb'\%o' % i, bytes([i & 0o377]))
```

```
1215 1224
```

```
1225 +                 with self.assertWarnsRegex(DeprecationWarning,
```

1226	+	r"invalid escape sequence '\\z'):
1227	+	self.assertEqual(decode(br'\x\z', 'ignore'), (b'\z', 4))
1228	+	with self.assertWarnsRegex(DeprecationWarning,
1229	+	r"invalid octal escape sequence '\\501'):
1230	+	self.assertEqual(decode(br'\x\501', 'ignore'), (b'A', 6))
1231	+	
1216	1232	def test_errors(self):
1217	1233	decode = codecs.escape_decode
1218	1234	self.assertRaises(ValueError, decode, br"\x")
		@@ -2661,24 +2677,40 @@ def test_escape_decode(self):
2661	2677	check(br"[\x410]", "[A0]")
2662	2678	check(br"\u20ac", "\u20ac")
2663	2679	check(br"\U0001d120", "\U0001d120")
2680	+	
2681	+	def test_decode_warnings(self):
2682	+	decode = codecs.unicode_escape_decode
2683	+	check = coding_checker(self, decode)
2664	2684	for i in range(97, 123):
2665	2685	b = bytes([i])
2666	2686	if b not in b'abfnrtuvx':
2667	-	with self.assertWarns(DeprecationWarning):
2687	+	with self.assertWarnsRegex(DeprecationWarning,
2688	+	r"invalid escape sequence '\\%c'" % i):
2668	2689	check(b"\\" + b, "\\" + chr(i))
2669	2690	if b.upper() not in b'UN':
2670	-	with self.assertWarns(DeprecationWarning):
2691	+	with self.assertWarnsRegex(DeprecationWarning,
2692	+	r"invalid escape sequence '\\%c'" % (i-32)):
2671	2693	check(b"\\" + b.upper(), "\\" + chr(i-32))
2672	-	with self.assertWarns(DeprecationWarning):
2694	+	with self.assertWarnsRegex(DeprecationWarning,
2695	+	r"invalid escape sequence '\\8'):
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"invalid escape sequence '\\\xfa') 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"invalid octal escape sequence '\\%o'" % i):
2680 2705                 check(rb'\%o' % i, chr(i))
2681 2706
2707 +             with self.assertWarnsRegex(DeprecationWarning,
2708 +                 r"invalid escape sequence '\\z'"):
2709 +                 self.assertEqual(decode(br'\x\z', 'ignore'), ('\z', 4))
2710 +             with self.assertWarnsRegex(DeprecationWarning,
2711 +                 r"invalid octal escape sequence '\\501'"):
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

...



@@ -1065,10 +1065,11 @@ _PyBytes_FormatEx(const char *format, Py_ssize_t
 format_len,

```

1065 1065     }
1066 1066
1067 1067     /* Unescape a backslash-escaped string. */
1068 - PyObject *_PyBytes_DecodeEscape(const char *s,
1068 + PyObject *_PyBytes_DecodeEscape2(const char *s,
1069 1069                                     Py_ssize_t len,
1070 1070                                     const char *errors,
1071 -                                     const char **first_invalid_escape)
1071 +                                     int *first_invalid_escape_char,
1072 +                                     const char **first_invalid_escape_ptr)
1072 1073     {
1073 1074         int c;
1074 1075         char *p;

```

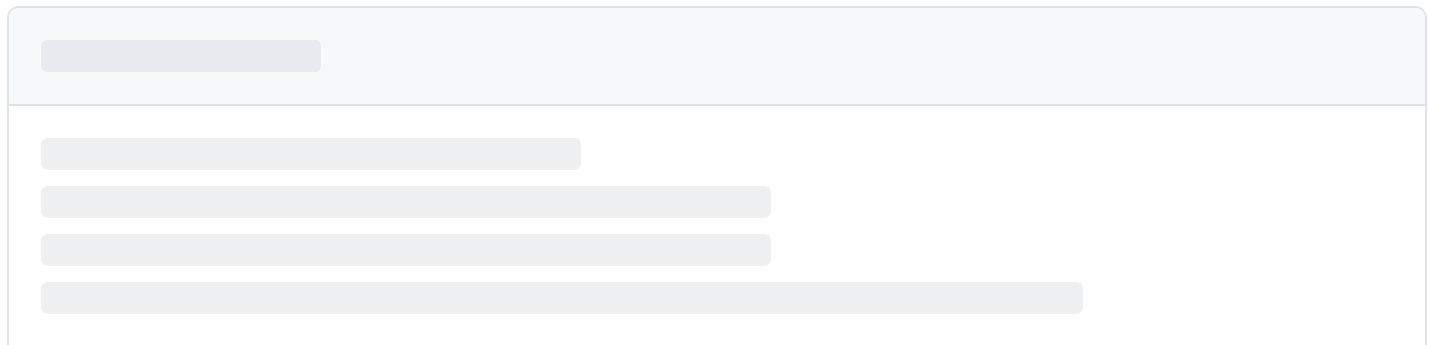
		@@ -1082,7 +1083,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1082	1083	return NULL;
1083	1084	writer.overallocate = 1;
1084	1085	
1085	-	*first_invalid_escape = NULL;
	1086	+ *first_invalid_escape_char = -1;
	1087	+ *first_invalid_escape_ptr = NULL;
1086	1088	
1087	1089	end = s + len;
1088	1090	while (s < end) {
		@@ -1120,9 +1122,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
		↑
1120	1122	c = (c<<3) + *s++ - '0';
1121	1123	}
1122	1124	if (c > 0377) {
1123	-	if (*first_invalid_escape == NULL) {
1124	-	*first_invalid_escape = s-3; /* Back up 3 chars, since
		we've
1125	-	already incremented s. */
	1125	+ if (*first_invalid_escape_char == -1) {
	1126	+ *first_invalid_escape_char = c;
	1127	+ /* Back up 3 chars, since we've already incremented s. */
	1128	+ *first_invalid_escape_ptr = s - 3;
1126	1129	}
1127	1130	}
1128	1131	*p++ = c;
		@@ -1163,9 +1166,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
		↑
1163	1166	break;
1164	1167	
1165	1168	default:
1166	-	if (*first_invalid_escape == NULL) {
1167	-	*first_invalid_escape = s-1; /* Back up one char, since we've
1168	-	already incremented s. */
	1169	+ if (*first_invalid_escape_char == -1) {
	1170	+ *first_invalid_escape_char = (unsigned char)s[-1];
	1171	+ /* Back up one char, since we've already incremented s. */
	1172	+ *first_invalid_escape_ptr = s - 1;
1169	1173	}
1170	1174	*p++ = '\\';

```

1171 1175          s--;
1172 1176      @@ -1179,23 +1183,37 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1173 1177
1174 1178          return NULL;
1175 1179      }
1176 1180
1177 1181
1178 1182 + // Export for binary compatibility.
1179 1183 + PyObject *_PyBytes_DecodeEscape(const char *s,
1180 1184 +         Py_ssize_t len,
1181 1185 +         const char *errors,
1182 1186 +         const char **first_invalid_escape)
1183 1187 + {
1184 1188 +     int first_invalid_escape_char;
1185 1189 +     return _PyBytes_DecodeEscape2(
1186 1190 +         s, len, errors,
1187 1191 +         &first_invalid_escape_char,
1188 1192 +         first_invalid_escape);
1189 1193 + }
1190 1194 +
1191 1195
1192 1196 PyObject *_PyBytes_DecodeEscape(const char *s,
1193 1197         Py_ssize_t len,
1194 1198         const char *errors,
1195 1199         Py_ssize_t Py_UNUSED(unicode),
1196 1200         const char *Py_UNUSED(recode_encoding))
1197 1201     {
1198 1202 -     const char* first_invalid_escape;
1199 1203 -     PyObject *result = _PyBytes_DecodeEscape(s, len, errors,
1200 1204 -         &first_invalid_escape);
1201 1205 +     int first_invalid_escape_char;
1202 1206 +     const char *first_invalid_escape_ptr;
1203 1207 +     PyObject *result = _PyBytes_DecodeEscape2(s, len, errors,
1204 1208 +         &first_invalid_escape_char,
1205 1209 +         &first_invalid_escape_ptr);
1206 1210
1207 1211     if (result == NULL)
1208 1212         return NULL;
1209 1213 -     if (first_invalid_escape != NULL) {
1210 1214 -         unsigned char c = *first_invalid_escape;
1211 1215 -         if ('4' <= c && c <= '7') {
1212 1216 +         if (first_invalid_escape_char != -1) {
1213 1217 +             if (first_invalid_escape_char > 0xff) {
1214 1218
1215 1219         if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,

```

1197	-	"invalid octal escape sequence '\\%.3s'",
1198	-	first_invalid_escape) < 0)
1215	+	"invalid octal escape sequence '\\%0'",
1216	+	first_invalid_escape_char) < 0)
1199	1217	{
1200	1218	Py_DECREF(result);
1201	1219	return NULL;
⋮	@@ -1204,7 +1222,7 @@	PyObject *PyBytes_DecodeEscape(const char *s,
1204	1222	else {
1205	1223	if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
1206	1224	"invalid escape sequence '\\%c'",
1207	-	c) < 0)
1225	+	first_invalid_escape_char) < 0)
1208	1226	{
1209	1227	Py_DECREF(result);
1210	1228	return NULL;
⋮		



Comments 0