

python / cpython Public

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

Commit 73b3040



serhiy-storchaka authored on Jun 2, 2025 · 17 / 18 · Verified

[3.11] [gh-133767](#): Fix use-after-free in the unicode-escape decoder with an error handler ([GH-129648](#)) ([GH-133944](#)) ([GH-134341](#))

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](#))
- (cherry picked from commit [6279eb8](#))
- (cherry picked from commit [a75953b](#))

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

3.11 (#98846, #134341) · v3.11.15 ... v3.11.13

1 parent [461ca2c](#) commit 73b3040

8 files changed

+197 -57

Top






Include/cpython


- bytesobject.h
- unicodeobject.h

Lib/test

- test_codeccallbacks.py
- test_codecs.py

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


 Search within code 

v Include/cpython/bytesobject.h 

```

↑... @@ -25,6 +25,10 @@ PyAPI_FUNC(PyObject*) _PyBytes_FromHex(
25 25     int use_bytearray);
26 26
27 27     /* Helper for PyBytes_DecodeEscape that detects invalid escape chars. */
28 + PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape2(const char *, Py_ssize_t,
29 +                                               const char *,
30 +                                               int *, const char **);
31 + // Export for binary compatibility.
28 32     PyAPI_FUNC(PyObject *) _PyBytes_DecodeEscape(const char *, Py_ssize_t,
29 33                                               const char *, const char **);
30 34

```

v Include/cpython/unicodeobject.h 

```

↑... @@ -914,6 +914,19 @@ PyAPI_FUNC(PyObject*)
_PyUnicode_DecodeUnicodeEscapeStateful(
914 914 );
915 915     /* Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
916 916     chars. */
917 + PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
918 +     const char *string, /* Unicode-Escape encoded string */
919 +     Py_ssize_t length, /* size of string */
920 +     const char *errors, /* error handling */
921 +     Py_ssize_t *consumed, /* bytes consumed */
922 +     int *first_invalid_escape_char, /* on return, if not -1, contain the first

```

```

923 +         invalid escaped char (<= 0xff) or
        invalid
924 +         octal escape (> 0xff) in string. */
925 +     const char **first_invalid_escape_ptr); /* on return, if not NULL, may
926 +         point to the first invalid escaped
927 +         char in string.
928 +         May be NULL if errors is not NULL. */
929 + // Export for binary compatibility.
917 930 PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
918 931     const char *string, /* Unicode-Escape encoded string */
919 932     Py_ssize_t length, /* size of string */

```



Lib/test/test_codeccallbacks.py



```

@@ -1124,7 +1124,7 @@ def test_bug828737(self):
1124 1124         text = 'abc<def>ghi'*n
1125 1125         text.translate(charmap)
1126 1126
1127 -     def test_mutating_decode_handler(self):
1127 +     def test_mutating_decode_handler(self):
1128 1128         baddata = [
1129 1129             ("ascii", b"\xff"),
1130 1130             ("utf-7", b"++"),
@@ -1159,6 +1159,42 @@ def mutating(exc):
1159 1159         for (encoding, data) in baddata:
1160 1160             self.assertEqual(data.decode(encoding, "test.mutating"),
                "\u4242")
1161 1161
1162 +     def test_mutating_decode_handler_unicode_escape(self):
1163 +         decode = codecs.unicode_escape_decode
1164 +         def mutating(exc):
1165 +             if isinstance(exc, UnicodeDecodeError):
1166 +                 r = data.get(exc.object[:exc.end])
1167 +                 if r is not None:
1168 +                     exc.object = r[0] + exc.object[exc.end:]
1169 +                 return ('\u0404', r[1])
1170 +             raise AssertionError("don't know how to handle %r" % exc)
1171 +
1172 +         codecs.register_error('test.mutating2', mutating)

```

```

1173 +         data = {
1174 +             br'\x0': (b'\\', 0),
1175 +             br'\x3': (b'xxx\\', 3),
1176 +             br'\x5': (b'x\\', 1),
1177 +         }
1178 +         def check(input, expected, msg):
1179 +             with self.assertWarns(DeprecationWarning) as cm:
1180 +                 self.assertEqual(decode(input, 'test.mutating2'), (expected,
1181 + len(input)))
1182 +                 self.assertIn(msg, str(cm.warning))
1183 +                 check(br'\x0n\z', '\u0404n\\z', r"invalid escape sequence '\z'")
1184 +                 check(br'\x0n\501', '\u0404n\u0141', r"invalid octal escape sequence
1185 + '\501'")
1186 +                 check(br'\x0z', '\u0404\\z', r"invalid escape sequence '\z'")
1187 +                 check(br'\x3n\zr', '\u0404n\\zr', r"invalid escape sequence '\z'")
1188 +                 check(br'\x3zr', '\u0404\\zr', r"invalid escape sequence '\z'")
1189 +                 check(br'\x3z5', '\u0404\\z5', r"invalid escape sequence '\z'")
1190 +                 check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r"invalid escape
1191 + sequence '\z'")
1192 +                 check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r"invalid escape
1193 + sequence '\z'")
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'")
1197 +                 check(br'\x5z', '\u0404\\z', r"invalid escape sequence '\z'")
1198 +                 check(memoryview(br'\x5zy')[:-1], '\u0404\\z', r"invalid escape
1199 + sequence '\z'")
1200 +
1201 + # issue32583
1202 + def test_crashing_decode_handler(self):
1203 +     # better generating one more character to fill the extra space slot

```



Lib/test/test_codec.py



@@ -1198,23 +1198,39 @@ def test_escape(self):

1198 1198 check(br"[\1010]", b"[A0]")

1199 1199 check(br"[\x41]", b"[A]")

```

1200 1200         check(br"[\x410]", b"[A0]")
1201 1201 +
1202 1202 +     def test_warnings(self):
1203 1203 +         decode = codecs.escape_decode
1204 1204 +         check = coding_checker(self, decode)
1205 1205         for i in range(97, 123):
1206 1206             b = bytes([i])
1207 1207             if b not in b'abfnrtvx':
1208 1208 -                 with self.assertWarns(DeprecationWarning):
1209 1209 +                 with self.assertWarnsRegex(DeprecationWarning,
1210 1210 +                     r"invalid escape sequence '\\%c'" % i):
1211 1211         check(b"\\\" + b, b"\\\" + b)
1212 1212 -                 with self.assertWarns(DeprecationWarning):
1213 1213 +                 with self.assertWarnsRegex(DeprecationWarning,
1214 1214 +                     r"invalid escape sequence '\\%c'" % (i-32)):
1215 1215         check(b"\\\" + b.upper(), b"\\\" + b.upper())
1216 1216 -                 with self.assertWarns(DeprecationWarning):
1217 1217 +                 with self.assertWarnsRegex(DeprecationWarning,
1218 1218 +                     r"invalid escape sequence '\\8'"):
1219 1219         check(br"\8", b"\\8")
1220 1220         with self.assertWarns(DeprecationWarning):
1221 1221         check(br"\9", b"\\9")
1222 1222 -                 with self.assertWarns(DeprecationWarning):
1223 1223 +                 with self.assertWarnsRegex(DeprecationWarning,
1224 1224 +                     r"invalid escape sequence '\\\xfa'") as cm:
1225 1225         check(b"\\\xfa", b"\\\xfa")
1226 1226         for i in range(0o400, 0o1000):
1227 1227 -                 with self.assertWarns(DeprecationWarning):
1228 1228 +                 with self.assertWarnsRegex(DeprecationWarning,
1229 1229 +                     r"invalid octal escape sequence '\\%o'" % i):
1230 1230         check(rb'\%o' % i, bytes([i & 0o377]))
1231 1231 +                 with self.assertWarnsRegex(DeprecationWarning,
1232 1232 +                     r"invalid octal escape sequence '\\501'"):
1233 1233 +                     self.assertEqual(decode(br'\x501', 'ignore'), (b'\z', 4))
1234 1234 +                 with self.assertWarnsRegex(DeprecationWarning,
1235 1235 +                     r"invalid octal escape sequence '\\501'"):
1236 1236 +                     self.assertEqual(decode(br'\x501', 'ignore'), (b'A', 6))
1237 1237         def test_errors(self):

```

```

1219 1235         decode = codecs.escape_decode
1220 1236         self.assertRaises(ValueError, decode, br"\x")
@@ -2487,24 +2503,40 @@ def test_escape_decode(self):
2487 2503         check(br"[\x410]", "[A0]")
2488 2504         check(br"\u20ac", "\u20ac")
2489 2505         check(br"\U0001d120", "\U0001d120")
2506 +
2507 +         def test_decode_warnings(self):
2508 +             decode = codecs.unicode_escape_decode
2509 +             check = coding_checker(self, decode)
2490 2510         for i in range(97, 123):
2491 2511             b = bytes([i])
2492 2512             if b not in b'abfnrtuvx':
2493 -                 with self.assertWarns(DeprecationWarning):
2513 +                 with self.assertWarnsRegex(DeprecationWarning,
2514 +                                             r"invalid escape sequence '\\%c'" % i):
2494 2515                     check(b"\" + b, "\"" + chr(i))
2495 2516                     if b.upper() not in b'UN':
2496 -                         with self.assertWarns(DeprecationWarning):
2517 +                         with self.assertWarnsRegex(DeprecationWarning,
2518 +                                                     r"invalid escape sequence '\\%c'" % (i-32)):
2497 2519                             check(b"\" + b.upper(), "\"" + chr(i-32))
2498 -                 with self.assertWarns(DeprecationWarning):
2520 +                 with self.assertWarnsRegex(DeprecationWarning,
2521 +                                             r"invalid escape sequence '\\8'"):
2499 2522                     check(br"\8", "\\8")
2500 2523                     with self.assertWarns(DeprecationWarning):
2501 2524                         check(br"\9", "\\9")
2502 -                 with self.assertWarns(DeprecationWarning):
2525 +                 with self.assertWarnsRegex(DeprecationWarning,
2526 +                                             r"invalid escape sequence '\\\xfa'") as cm:
2503 2527                     check(b"\\\xfa", "\\xfa")
2504 2528                     for i in range(0o400, 0o1000):
2505 -                         with self.assertWarns(DeprecationWarning):
2529 +                         with self.assertWarnsRegex(DeprecationWarning,
2530 +                                                     r"invalid octal escape sequence '\\%o'" % i):
2506 2531                             check(rb'\%o' % i, chr(i))
2507 2532
2533 +                 with self.assertWarnsRegex(DeprecationWarning,

```

```

2534 +         r"invalid escape sequence '\\z"):
2535 +         self.assertEqual(decode(br'\x\z', 'ignore'), ('\\z', 4))
2536 +         with self.assertWarnsRegex(DeprecationWarning,
2537 +         r"invalid octal escape sequence '\\501"):
2538 +             self.assertEqual(decode(br'\x\501', 'ignore'), ('\u0141', 6))
2539 +
2508 2540         def test_decode_errors(self):
2509 2541             decode = codecs.unicode_escape_decode
2510 2542             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

```

@@ -1057,10 +1057,11 @@ _PyBytes_FormatEx(const char *format, Py_ssize_t
format_len,
1057 1057     }
1058 1058
1059 1059     /* Unescape a backslash-escaped string. */
1060 - PyObject *_PyBytes_DecodeEscape(const char *s,
1060 + PyObject *_PyBytes_DecodeEscape2(const char *s,
1061 1061                                     Py_ssize_t len,
1062 1062                                     const char *errors,
1063 -                                     const char **first_invalid_escape)
1063 +                                     int *first_invalid_escape_char,
1064 +                                     const char **first_invalid_escape_ptr)
1064 1065     {
1065 1066         int c;
1066 1067         char *p;
@@ -1074,7 +1075,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1074 1075         return NULL;
1075 1076         writer.overallocate = 1;
1076 1077
1077 -         *first_invalid_escape = NULL;
1078 +         *first_invalid_escape_char = -1;
1079 +         *first_invalid_escape_ptr = NULL;

```

```

1078 1080
1079 1081     end = s + len;
1080 1082     while (s < end) {
    ↓
    ↑
@@ -1112,9 +1114,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1112 1114         c = (c<<3) + *s++ - '0';
1113 1115     }
1114 1116     if (c > 0377) {
1115 -         if (*first_invalid_escape == NULL) {
1116 -             *first_invalid_escape = s-3; /* Back up 3 chars, since
we've
1117 -                 already incremented s. */
1117 +         if (*first_invalid_escape_char == -1) {
1118 +             *first_invalid_escape_char = c;
1119 +             /* Back up 3 chars, since we've already incremented s. */
1120 +             *first_invalid_escape_ptr = s - 3;
1118 1121     }
1119 1122     }
1120 1123     *p++ = c;
    ↓
    ↑
@@ -1155,9 +1158,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1155 1158         break;
1156 1159
1157 1160     default:
1158 -         if (*first_invalid_escape == NULL) {
1159 -             *first_invalid_escape = s-1; /* Back up one char, since we've
1160 -                 already incremented s. */
1161 +         if (*first_invalid_escape_char == -1) {
1162 +             *first_invalid_escape_char = (unsigned char)s[-1];
1163 +             /* Back up one char, since we've already incremented s. */
1164 +             *first_invalid_escape_ptr = s - 1;
1161 1165     }
1162 1166     *p++ = '\\';
1163 1167     s--;
    ↕
@@ -1171,23 +1175,39 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1171 1175     return NULL;
1172 1176 }
1173 1177
1178 + // Export for binary compatibility.
1179 + PyObject *_PyBytes_DecodeEscape(const char *s,

```

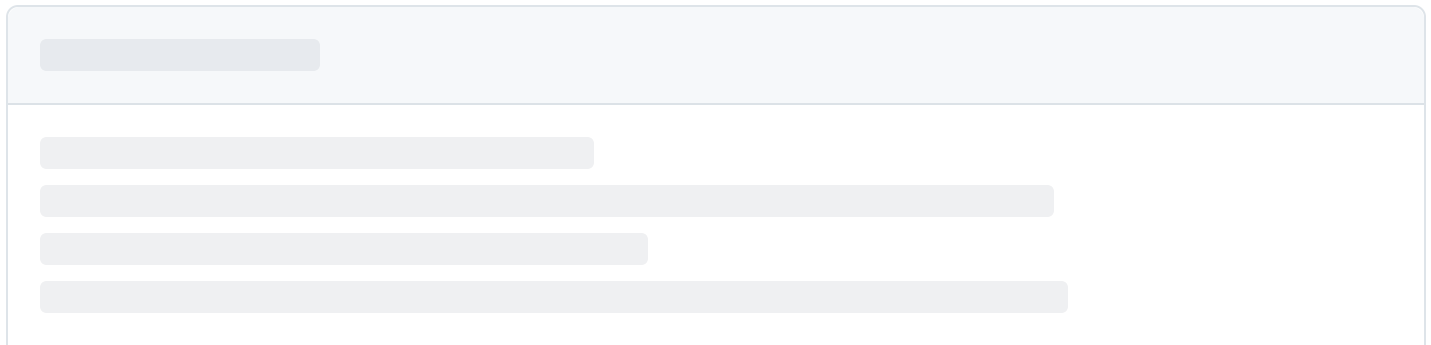
```

1180 +         Py_ssize_t len,
1181 +         const char *errors,
1182 +         const char **first_invalid_escape)
1183 + {
1184 +     int first_invalid_escape_char;
1185 +     return _PyBytes_DecodeEscape2(
1186 +         s, len, errors,
1187 +         &first_invalid_escape_char,
1188 +         first_invalid_escape);
1189 + }
1190 +
1174 1191     PyObject *PyBytes_DecodeEscape(const char *s,
1175 1192         Py_ssize_t len,
1176 1193         const char *errors,
1177 1194         Py_ssize_t Py_UNUSED(unicode),
1178 1195         const char *Py_UNUSED(recode_encoding))
1179 1196     {
1180 -         const char* first_invalid_escape;
1181 -         PyObject *result = _PyBytes_DecodeEscape(s, len, errors,
1182 -             &first_invalid_escape);
1197 +         int first_invalid_escape_char;
1198 +         const char *first_invalid_escape_ptr;
1199 +         PyObject *result = _PyBytes_DecodeEscape2(s, len, errors,
1200 +             &first_invalid_escape_char,
1201 +             &first_invalid_escape_ptr);
1202     if (result == NULL)
1203         return NULL;
1204 -     if (first_invalid_escape != NULL) {
1205 -         unsigned char c = *first_invalid_escape;
1206 -         if ('4' <= c && c <= '7') {
1207 +         if (first_invalid_escape_char != -1) {
1208 +             if (first_invalid_escape_char > 0xff) {
1209 +                 char buf[12] = "";
1210 +                 snprintf(buf, sizeof buf, "%0", first_invalid_escape_char);
1211 +                 if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
1212 +                     "invalid octal escape sequence '\\%.3s'",
1213 +                     first_invalid_escape) < 0)
1214 +                     "invalid octal escape sequence '\\%s'",
1215 +                     buf) < 0)
1216     {
1191 1211         {

```

```
1192 1212         Py_DECREF(result);
1193 1213         return NULL;
@@ -1196,7 +1216,7 @@ PyObject *PyBytes_DecodeEscape(const char *s,
1196 1216         else {
1197 1217             if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
1198 1218                 "invalid escape sequence '\\%c'",
1199 -                c) < 0)
+                first_invalid_escape_char) < 0)
1200 1220             {
1201 1221                 Py_DECREF(result);
1202 1222                 return NULL;

```



Comments 0