

python / cpython Public

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

# Commit 9f69a58



serhiy-storchaka authored on May 12, 2025 · 24 / 38 · Verified

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

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()`.

main (#129648) · v3.15.0a8 ... v3.15.0a1

1 parent [734e15b](#) commit 9f69a58


**8 files changed** +160 -63 lines changed

Top


Filter files...

- ▼ Include/internal
  - `pycore_bytesobject.h`
  - `pycore_unicodeobject.h`
- ▼ Lib/test
  - `test_codeccallbacks.py`
  - `test_codecscs.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


 **8 files changed** **+160 -63** lines changed


 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
        invalid
149 +         octal escape (> 0xff) in string. */
150 +     const char **first_invalid_escape_ptr); /* on return, if not NULL, may
151 +         point to the first invalid escaped
152 +         char in string.
153 +         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 + 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"),
                "\u4242")
1162 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')
1198 +                 check(br'\x5n\z', '\u0404\n\\z', r'"z" is an invalid escape
1199 + sequence')
1200 +                 check(br'\x5n\501', '\u0404\n\u0141', r'"501" is an invalid octal
1201 + 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
1199 +         escape sequence')
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\_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'"\\%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 +                     with self.assertWarnsRegex(DeprecationWarning,
1218 +                     r'"\\xfa" is an invalid escape sequence') as cm:
1211 1219                         check(b"\\xfa", b"\\xfa")
1212 1220                         for i in range(0o400, 0o1000):

```

1213	-	<code>with self.assertWarns(DeprecationWarning):</code>
1221	+	<code>with self.assertWarnsRegex(DeprecationWarning,</code>
1222	+	<code>    r'"\%0" is an invalid octal escape sequence' % i):</code>
1214	1223	<code>    check(rb'\%0' % i, bytes([i &amp; 0o377]))</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'\x\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	-	<code>with self.assertWarns(DeprecationWarning):</code>
2694	+	<code>with self.assertWarnsRegex(DeprecationWarning,</code>
2695	+	<code>    r'"\8" is an invalid escape sequence'):</code>
2673	2696	<code>    check(br"\8", "\8")</code>

```

2674 2697         with self.assertWarns(DeprecationWarning):
2675 2698             check(br"\9", "\\9")
2676 -         with self.assertWarns(DeprecationWarning):
2699 +         with self.assertWarnsRegex(DeprecationWarning,
2700 +             r'"\x{fa}" is an invalid escape sequence') as cm:
2677 2701             check(b"\x{fa}", "\\x{fa}")
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	-	<code>const char **first_invalid_escape)</code>
1081	+	<code>int *first_invalid_escape_char,</code>
1082	+	<code>const char **first_invalid_escape_ptr)</code>
1082	1083	<code>{</code>
1083	1084	<code>int c;</code>
1084	1085	<code>char *p;</code>
		<code>@@ -1092,7 +1093,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,</code>
1092	1093	<code>return NULL;</code>
1093	1094	<code>writer.overallocate = 1;</code>
1094	1095	
1095	-	<code>*first_invalid_escape = NULL;</code>
1096	+	<code>*first_invalid_escape_char = -1;</code>
1097	+	<code>*first_invalid_escape_ptr = NULL;</code>
1096	1098	
1097	1099	<code>end = s + len;</code>
1098	1100	<code>while (s &lt; end) {</code>
		<code>@@ -1130,9 +1132,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,</code>
1130	1132	<code>c = (c&lt;&lt;3) + *s++ - '0';</code>
1131	1133	<code>}</code>
1132	1134	<code>if (c &gt; 0377) {</code>
1133	-	<code>if (*first_invalid_escape == NULL) {</code>
1134	-	<code>*first_invalid_escape = s-3; /* Back up 3 chars, since</code>
		<code>we've</code>
1135	-	<code>already incremented s. */</code>
1135	+	<code>if (*first_invalid_escape_char == -1) {</code>
1136	+	<code>*first_invalid_escape_char = c;</code>
1137	+	<code>/* Back up 3 chars, since we've already incremented s. */</code>
1138	+	<code>*first_invalid_escape_ptr = s - 3;</code>
1136	1139	<code>}</code>
1137	1140	<code>}</code>
1138	1141	<code>*p++ = c;</code>
		<code>@@ -1173,9 +1176,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,</code>
1173	1176	<code>break;</code>
1174	1177	
1175	1178	<code>default:</code>
1176	-	<code>if (*first_invalid_escape == NULL) {</code>
1177	-	<code>*first_invalid_escape = s-1; /* Back up one char, since we've</code>
1178	-	<code>already incremented s. */</code>

```

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;

```

↓

▼ Objects/unicodeobject.c

```

↑... @@ -6596,21 +6596,24 @@ _PyUnicode_GetNameCAPI(void)
6596 6596 /* --- Unicode Escape Codec -----
*/
6597 6597
6598 6598 PyObject *
6599 - _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
6599 + _PyUnicode_DecodeUnicodeEscapeInternal2(const char *s,
6600 6600         Py_ssize_t size,
6601 6601         const char *errors,
6602 6602         Py_ssize_t *consumed,
6603 -         const char **first_invalid_escape)
6603 +         int *first_invalid_escape_char,
6604 +         const char **first_invalid_escape_ptr)
6604 6605     {
6605 6606         const char *starts = s;
6607 +         const char *initial_starts = starts;
6606 6608         _PyUnicodeWriter writer;
6607 6609         const char *end;
6608 6610         PyObject *errorHandler = NULL;
6609 6611         PyObject *exc = NULL;
6610 6612         _PyUnicode_Name_CAPI *ucnhash_capi;
6611 6613
6612 6614         // so we can remember if we've seen an invalid escape char or not
6613 -         *first_invalid_escape = NULL;
6615 +         *first_invalid_escape_char = -1;
6616 +         *first_invalid_escape_ptr = NULL;
6614 6617
6615 6618         if (size == 0) {
6616 6619             if (consumed) {

```

		@@ -6698,9 +6701,12 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
		*s,
6698	6701	}
6699	6702	}
6700	6703	if (ch > 0377) {
6701	-	if (*first_invalid_escape == NULL) {
6702	-	*first_invalid_escape = s-3; /* Back up 3 chars, since
		we've
6703	-	already incremented s. */
6704	+	if (*first_invalid_escape_char == -1) {
6705	+	*first_invalid_escape_char = ch;
6706	+	if (starts == initial_starts) {
6707	+	/* Back up 3 chars, since we've already incremented
		s. */
6708	+	*first_invalid_escape_ptr = s - 3;
6709	+	}
6704	6710	}
6705	6711	}
6706	6712	WRITE_CHAR(ch);
		@@ -6795,9 +6801,12 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
		*s,
6795	6801	goto error;
6796	6802	
6797	6803	default:
6798	-	if (*first_invalid_escape == NULL) {
6799	-	*first_invalid_escape = s-1; /* Back up one char, since we've
6800	-	already incremented s. */
6804	+	if (*first_invalid_escape_char == -1) {
6805	+	*first_invalid_escape_char = c;
6806	+	if (starts == initial_starts) {
6807	+	/* Back up one char, since we've already incremented s.
		*/
6808	+	*first_invalid_escape_ptr = s - 1;
6809	+	}
6801	6810	}
6802	6811	WRITE_ASCII_CHAR('\\');
6803	6812	WRITE_CHAR(c);
		@@ -6842,19 +6851,20 @@ _PyUnicode_DecodeUnicodeEscapeStateful(const char
		*s,
6842	6851	const char *errors,

```

6843 6852 Py_ssize_t *consumed)
6844 6853 {
6845 -   const char *first_invalid_escape;
6846 -   PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal(s, size,
        errors,
6854 +   int first_invalid_escape_char;
6855 +   const char *first_invalid_escape_ptr;
6856 +   PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal2(s, size,
        errors,
6847 6857                                     consumed,
6848 -                                     &first_invalid_escape);
6858 +
        &first_invalid_escape_char,
6859 +
        &first_invalid_escape_ptr);
6849 6860     if (result == NULL)
6850 6861         return NULL;
6851 -   if (first_invalid_escape != NULL) {
6852 -       unsigned char c = *first_invalid_escape;
6853 -       if ('4' <= c && c <= '7') {
6862 +   if (first_invalid_escape_char != -1) {
6863 +       if (first_invalid_escape_char > 0xff) {
6854 6864         if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
6855 -                               "\"\\%0.3s\" is an invalid octal escape
        sequence. "
6865 +                               "\"\\%0\" is an invalid octal escape
        sequence. "
6856 6866         "Such sequences will not work in the future.
        ",
6857 -                               first_invalid_escape) < 0)
6867 +                               first_invalid_escape_char) < 0)
6858 6868     {
6859 6869         Py_DECREF(result);
6860 6870         return NULL;
@@ -6864,7 +6874,7 @@ _PyUnicode_DecodeUnicodeEscapeStateful(const char
*s,
6864 6874         if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
6865 6875         "\"\\%c\" is an invalid escape sequence. "
6866 6876         "Such sequences will not work in the future.
        ",

```

```

6867 - c) < 0)
6877 + first_invalid_escape_char) < 0)
6868 6878 {
6869 6879     Py_DECREF(result);
6870 6880     return NULL;

```

Parser/string\_parser.c

```

@@ -196,15 +196,18 @@ decode_unicode_with_escapes(Parser *parser, const char
*s, size_t len, Token *t)
196 196     len = (size_t)(p - buf);
197 197     s = buf;
198 198
199 -     const char *first_invalid_escape;
200 -     v = _PyUnicode_DecodeUnicodeEscapeInternal(s, (Py_ssize_t)len, NULL, NULL,
&first_invalid_escape);
199 +     int first_invalid_escape_char;
200 +     const char *first_invalid_escape_ptr;
201 +     v = _PyUnicode_DecodeUnicodeEscapeInternal2(s, (Py_ssize_t)len, NULL, NULL,
202 +                                                 &first_invalid_escape_char,
203 +                                                 &first_invalid_escape_ptr);
201 204
202 205     // HACK: later we can simply pass the line no, since we don't preserve the
tokens
203 206     // when we are decoding the string but we preserve the line numbers.
204 -     if (v != NULL && first_invalid_escape != NULL && t != NULL) {
205 -         if (warn_invalid_escape_sequence(parser, s, first_invalid_escape, t) <
0) {
206 -             /* We have not decref u before because first_invalid_escape points
inside u. */
207 +         if (v != NULL && first_invalid_escape_ptr != NULL && t != NULL) {
208 +             if (warn_invalid_escape_sequence(parser, s, first_invalid_escape_ptr,
t) < 0) {
209 +                 /* We have not decref u before because first_invalid_escape_ptr
points inside u. */
208 211         Py_XDECREF(u);
209 212         Py_DECREF(v);
210 213         return NULL;
@@ -217,14 +220,17 @@ decode_unicode_with_escapes(Parser *parser, const char
*s, size_t len, Token *t)

```

```
217 220 static PyObject *
218 221 decode_bytes_with_escapes(Parser *p, const char *s, Py_ssize_t len, Token *t)
219 222 {
220 -     const char *first_invalid_escape;
221 -     PyObject *result = _PyBytes_DecodeEscape(s, len, NULL,
222 +     int first_invalid_escape_char;
223 +     const char *first_invalid_escape_ptr;
224 +     PyObject *result = _PyBytes_DecodeEscape2(s, len, NULL,
225 +     &first_invalid_escape_char,
226 +     &first_invalid_escape_ptr);
227 +
228     if (result == NULL) {
229         return NULL;
230     }
231
232 -     if (first_invalid_escape != NULL) {
233 -         if (warn_invalid_escape_sequence(p, s, first_invalid_escape, t) < 0) {
234 +     if (first_invalid_escape_ptr != NULL) {
235 +         if (warn_invalid_escape_sequence(p, s, first_invalid_escape_ptr, t) <
236 +         0) {
237
238         Py_DECREF(result);
239         return NULL;
240     }
241 }
```



Comments 0