Commit ab9893c

serhiy-storchaka authored on Jun 2, 2025 · ✕ 10 / 11 · Verified

```
[3.10] gh-133767: Fix use-after-free in the unicode-escape decoder with an error
handler (GH-129648) (GH-133944) (GH-134345)

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 became 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)
(cherry picked from commit 0c33e5b)

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

3.10 (#134345) · v3.10.20 ••• v3.10.18

1 parent f85e71a commit ab9893c

8 files changed +164 -41 lines changed

Top

Filter files…

- Include/cpython
  - bytesobject.h
  - 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

8 files changed +164 -41 lines changed

Search within code

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
```

Include/cpython/unicodeobject.h

```
@@ -844,6 +844,19 @@ PyAPI_FUNC(PyObject*)
_PyUnicode_DecodeUnicodeEscapeStateful(
844  844
845  845  /* Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
846  846     chars. */
    847  + PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
    848  +     const char *string,     /* Unicode-Escape encoded string */
    849  +     Py_ssize_t length,      /* size of string */
    850  +     const char *errors,     /* error handling */
    851  +     Py_ssize_t *consumed,   /* bytes consumed */
    852  +     int *first_invalid_escape_char, /* on return, if not -1, contain the first
    853  +                                        invalid escaped char (<= 0xff) or
invalid
```

```
+                                        octal escape (> 0xff) in string. */
+     const char **first_invalid_escape_ptr); /* on return, if not NULL, may
+                                             point to the first invalid escaped
+                                             char in string.
+                                             May be NULL if errors is not NULL. */
+ // Export for binary compatibility.
 PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
         const char *string,     /* Unicode-Escape encoded string */
         Py_ssize_t length,      /* size of string */
```

Lib/test/test_codeccallbacks.py

```
@@ -1124,7 +1124,7 @@ def test_bug828737(self):
             text = 'abc<def>ghi'*n
             text.translate(charmap)

-    def test_mutatingdecodehandler(self):
+    def test_mutating_decode_handler(self):
         baddata = [
             ("ascii", b"\xff"),
             ("utf-7", b"++"),
@@ -1159,6 +1159,40 @@ def mutating(exc):
             for (encoding, data) in baddata:
                 self.assertEqual(data.decode(encoding, "test.mutating"),
"\u4242")

+    def test_mutating_decode_handler_unicode_escape(self):
+        decode = codecs.unicode_escape_decode
+        def mutating(exc):
+            if isinstance(exc, UnicodeDecodeError):
+                r = data.get(exc.object[:exc.end])
+                if r is not None:
+                    exc.object = r[0] + exc.object[exc.end:]
+                    return ('\u0404', r[1])
+            raise AssertionError("don't know how to handle %r" % exc)
+
+        codecs.register_error('test.mutating2', mutating)
+        data = {
+            br'\x0': (b'\\', 0),
```

```
+            br'\x3': (b'xxx\\', 3),
+            br'\x5': (b'x\\', 1),
+        }
+        def check(input, expected, msg):
+            with self.assertWarns(DeprecationWarning) as cm:
+                self.assertEqual(decode(input, 'test.mutating2'), (expected, len(input)))
+            self.assertIn(msg, str(cm.warning))
+
+        check(br'\x0n\z', '\u0404\n\\z', r"invalid escape sequence '\z'")
+        check(br'\x0z', '\u0404\\z', r"invalid escape sequence '\z'")
+
+        check(br'\x3n\zr', '\u0404\n\\zr', r"invalid escape sequence '\z'")
+        check(br'\x3zr', '\u0404\\zr', r"invalid escape sequence '\z'")
+        check(br'\x3z5', '\u0404\\z5', r"invalid escape sequence '\z'")
+        check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r"invalid escape sequence '\z'")
+        check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r"invalid escape sequence '\z'")
+
+        check(br'\x5n\z', '\u0404\n\\z', r"invalid escape sequence '\z'")
+        check(br'\x5z', '\u0404\\z', r"invalid escape sequence '\z'")
+        check(memoryview(br'\x5zy')[:-1], '\u0404\\z', r"invalid escape sequence '\z'")
+
     # issue32583
     def test_crashing_decode_handler(self):
         # better generating one more character to fill the extra space slot
```

Lib/test/test_codecs.py

```
@@ -1181,20 +1181,32 @@ def test_escape(self):
            check(br"[\501]", b"[A]")
            check(br"[\x41]", b"[A]")
            check(br"[\x410]", b"[A0]")
+
+    def test_warnings(self):
+        decode = codecs.escape_decode
+        check = coding_checker(self, decode)
            for i in range(97, 123):
```

```
1185  1189                b = bytes([i])
1186  1190                if b not in b'abfnrtvx':
1187        -                 with self.assertWarns(DeprecationWarning):
      1191  +                 with self.assertWarnsRegex(DeprecationWarning,
      1192  +                         r"invalid escape sequence '\\%c'" % i):
1188  1193                      check(b"\\" + b, b"\\" + b)
1189        -             with self.assertWarns(DeprecationWarning):
      1194  +             with self.assertWarnsRegex(DeprecationWarning,
      1195  +                     r"invalid escape sequence '\\%c'" % (i-32)):
1190  1196                  check(b"\\" + b.upper(), b"\\" + b.upper())
1191        -         with self.assertWarns(DeprecationWarning):
      1197  +         with self.assertWarnsRegex(DeprecationWarning,
      1198  +                 r"invalid escape sequence '\\8'"):
1192  1199              check(br"\8", b"\\8")
1193  1200          with self.assertWarns(DeprecationWarning):
1194  1201              check(br"\9", b"\\9")
1195        -         with self.assertWarns(DeprecationWarning):
      1202  +         with self.assertWarnsRegex(DeprecationWarning,
      1203  +                 r"invalid escape sequence '\\\xfa'") as cm:
1196  1204              check(b"\\\xfa", b"\\\xfa")
1197  1205
      1206  +         with self.assertWarnsRegex(DeprecationWarning,
      1207  +                 r"invalid escape sequence '\\z'"):
      1208  +             self.assertEqual(decode(br'\x\z', 'ignore'), (b'\\z', 4))
      1209  +
1198  1210      def test_errors(self):
1199  1211          decode = codecs.escape_decode
1200  1212          self.assertRaises(ValueError, decode, br"\x")
          @@ -2408,20 +2420,31 @@ def test_escape_decode(self):
2408  2420          check(br"[\x410]", "[A0]")
2409  2421          check(br"\u20ac", "\u20ac")
2410  2422          check(br"\U0001d120", "\U0001d120")
      2423  +
      2424  +     def test_decode_warnings(self):
      2425  +         decode = codecs.unicode_escape_decode
      2426  +         check = coding_checker(self, decode)
2411  2427          for i in range(97, 123):
2412  2428              b = bytes([i])
2413  2429              if b not in b'abfnrtuvx':
```

```
2414      -                with self.assertWarns(DeprecationWarning):
      2430 +                with self.assertWarnsRegex(DeprecationWarning,
      2431 +                        r"invalid escape sequence '\\%c'" % i):
2415  2432                      check(b"\\" + b, "\\" + chr(i))
2416  2433                  if b.upper() not in b'UN':
2417      -                with self.assertWarns(DeprecationWarning):
      2434 +                with self.assertWarnsRegex(DeprecationWarning,
      2435 +                        r"invalid escape sequence '\\%c'" % (i-32)):
2418  2436                      check(b"\\" + b.upper(), "\\" + chr(i-32))
2419      -            with self.assertWarns(DeprecationWarning):
      2437 +            with self.assertWarnsRegex(DeprecationWarning,
      2438 +                    r"invalid escape sequence '\\8'"):
2420  2439                check(br"\8", "\\8")
2421  2440            with self.assertWarns(DeprecationWarning):
2422  2441                check(br"\9", "\\9")
2423      -            with self.assertWarns(DeprecationWarning):
      2442 +            with self.assertWarnsRegex(DeprecationWarning,
      2443 +                    r"invalid escape sequence '\\\xfa'") as cm:
2424  2444                check(b"\\\xfa", "\\\xfa")
      2445 +            with self.assertWarnsRegex(DeprecationWarning,
      2446 +                    r"invalid escape sequence '\\z'"):
      2447 +                self.assertEqual(decode(br'\x\z', 'ignore'), ('\\z', 4))
2425  2448
2426  2449        def test_decode_errors(self):
2427  2450            decode = codecs.unicode_escape_decode
```

...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

```
      @@ -1089,10 +1089,11 @@ _PyBytes_FormatEx(const char *format, Py_ssize_t
      format_len,
1089  1089      }
1090  1090
1091  1091      /* Unescape a backslash-escaped string. */
1092      - PyObject *_PyBytes_DecodeEscape(const char *s,
```

```
                 +PyObject *_PyBytes_DecodeEscape2(const char *s,
1093  1093                                         Py_ssize_t len,
1094  1094                                         const char *errors,
1095        -                                      const char **first_invalid_escape)
      1095  +                                      int *first_invalid_escape_char,
      1096  +                                      const char **first_invalid_escape_ptr)
1096  1097    {
1097  1098        int c;
1098  1099        char *p;
            @@ -1106,7 +1107,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1106  1107            return NULL;
1107  1108        writer.overallocate = 1;
1108  1109
1109        -     *first_invalid_escape = NULL;
      1110  +     *first_invalid_escape_char = -1;
      1111  +     *first_invalid_escape_ptr = NULL;
1110  1112
1111  1113        end = s + len;
1112  1114        while (s < end) {
            @@ -1181,9 +1183,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1181  1183                break;
1182  1184
1183  1185            default:
1184        -           if (*first_invalid_escape == NULL) {
1185        -               *first_invalid_escape = s-1; /* Back up one char, since we've
1186        -                                               already incremented s. */
      1186  +           if (*first_invalid_escape_char == -1) {
      1187  +               *first_invalid_escape_char = (unsigned char)s[-1];
      1188  +               /* Back up one char, since we've already incremented s. */
      1189  +               *first_invalid_escape_ptr = s - 1;
1187  1190                }
1188  1191                *p++ = '\\';
1189  1192                s--;
            @@ -1197,21 +1200,36 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1197  1200        return NULL;
1198  1201    }
1199  1202
      1203  +// Export for binary compatibility.
      1204  +PyObject *_PyBytes_DecodeEscape(const char *s,
```

```
        1205 +                                  Py_ssize_t len,
        1206 +                                  const char *errors,
        1207 +                                  const char **first_invalid_escape)
        1208 + {
        1209 +     int first_invalid_escape_char;
        1210 +     return _PyBytes_DecodeEscape2(
        1211 +             s, len, errors,
        1212 +             &first_invalid_escape_char,
        1213 +             first_invalid_escape);
        1214 + }
        1215 +
1200    1216   PyObject *PyBytes_DecodeEscape(const char *s,
1201    1217                                  Py_ssize_t len,
1202    1218                                  const char *errors,
1203    1219                                  Py_ssize_t Py_UNUSED(unicode),
1204    1220                                  const char *Py_UNUSED(recode_encoding))
1205    1221   {
1206          -     const char* first_invalid_escape;
1207          -     PyObject *result = _PyBytes_DecodeEscape(s, len, errors,
1208          -                                              &first_invalid_escape);
        1222 +     int first_invalid_escape_char;
        1223 +     const char *first_invalid_escape_ptr;
        1224 +     PyObject *result = _PyBytes_DecodeEscape2(s, len, errors,
        1225 +                                               &first_invalid_escape_char,
        1226 +                                               &first_invalid_escape_ptr);
1209    1227       if (result == NULL)
1210    1228           return NULL;
1211          -     if (first_invalid_escape != NULL) {
        1229 +     if (first_invalid_escape_char != -1) {
1212    1230           if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
1213    1231                                "invalid escape sequence '\\%c'",
1214          -                              (unsigned char)*first_invalid_escape) < 0) {
        1232 +                              first_invalid_escape_char) < 0) {
1215    1233               Py_DECREF(result);
1216    1234               return NULL;
1217    1235           }
```

Objects/unicodeobject.c

```
@@ -6432,20 +6432,23 @@ PyUnicode_AsUTF16String(PyObject *unicode)
```

```
6432  6432    static _PyUnicode_Name_CAPI *ucnhash_capi = NULL;
6433  6433
6434  6434    PyObject *
6435        -   _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
      6435  +   _PyUnicode_DecodeUnicodeEscapeInternal2(const char *s,
6436  6436                                            Py_ssize_t size,
6437  6437                                            const char *errors,
6438  6438                                            Py_ssize_t *consumed,
6439        -                                       const char **first_invalid_escape)
      6439  +                                       int *first_invalid_escape_char,
      6440  +                                       const char **first_invalid_escape_ptr)
6440  6441    {
6441  6442        const char *starts = s;
      6443  +     const char *initial_starts = starts;
6442  6444        _PyUnicodeWriter writer;
6443  6445        const char *end;
6444  6446        PyObject *errorHandler = NULL;
6445  6447        PyObject *exc = NULL;
6446  6448
6447  6449        // so we can remember if we've seen an invalid escape char or not
6448        -     *first_invalid_escape = NULL;
      6450  +     *first_invalid_escape_char = -1;
      6451  +     *first_invalid_escape_ptr = NULL;
6449  6452
6450  6453        if (size == 0) {
6451  6454            if (consumed) {
          @@ -6628,9 +6631,12 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
          *s,
6628  6631                goto error;
6629  6632
6630  6633            default:
6631        -               if (*first_invalid_escape == NULL) {
6632        -                   *first_invalid_escape = s-1; /* Back up one char, since we've
6633        -                                                   already incremented s. */
      6634  +               if (*first_invalid_escape_char == -1) {
      6635  +                   *first_invalid_escape_char = c;
      6636  +                   if (starts == initial_starts) {
      6637  +                       /* Back up one char, since we've already incremented s.
          */
      6638  +                       *first_invalid_escape_ptr = s - 1;
```

```
                6639 +                 }
6634            6640                   }
6635            6641                   WRITE_ASCII_CHAR('\\');
6636            6642                   WRITE_CHAR(c);
                     @@ -6669,22 +6675,39 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
                     *s,
6669            6675       return NULL;
6670            6676   }
6671            6677
                6678 + // Export for binary compatibility.
                6679 + PyObject *
                6680 + _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
                6681 +                                Py_ssize_t size,
                6682 +                                const char *errors,
                6683 +                                Py_ssize_t *consumed,
                6684 +                                const char **first_invalid_escape)
                6685 + {
                6686 +     int first_invalid_escape_char;
                6687 +     return _PyUnicode_DecodeUnicodeEscapeInternal2(
                6688 +             s, size, errors, consumed,
                6689 +             &first_invalid_escape_char,
                6690 +             first_invalid_escape);
                6691 + }
                6692 +
6672            6693   PyObject *
6673            6694   _PyUnicode_DecodeUnicodeEscapeStateful(const char *s,
6674            6695                                Py_ssize_t size,
6675            6696                                const char *errors,
6676            6697                                Py_ssize_t *consumed)
6677            6698   {
6678                 -     const char *first_invalid_escape;
6679                 -     PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal(s, size,
                     errors,
                6699 +     int first_invalid_escape_char;
                6700 +     const char *first_invalid_escape_ptr;
                6701 +     PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal2(s, size,
                     errors,
6680            6702                                                    consumed,
6681                 -                                                  &first_invalid_escape);
```

```
                6703 +
                       &first_invalid_escape_char,
                6704 +
                       &first_invalid_escape_ptr);
6682            6705        if (result == NULL)
6683            6706            return NULL;
6684            -       if (first_invalid_escape != NULL) {
                6707 +      if (first_invalid_escape_char != -1) {
6685            6708            if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
6686            6709                                 "invalid escape sequence '\\%c'",
6687            -                                    (unsigned char)*first_invalid_escape) < 0) {
                6710 +                                    first_invalid_escape_char) < 0) {
6688            6711                Py_DECREF(result);
6689            6712                return NULL;
6690            6713            }
```

Parser/string_parser.c

```
                @@ -114,12 +114,15 @@ decode_unicode_with_escapes(Parser *parser, const char
                *s, size_t len, Token *t)
114             114         len = p - buf;
115             115         s = buf;
116             116
117             -       const char *first_invalid_escape;
118             -       v = _PyUnicode_DecodeUnicodeEscapeInternal(s, len, NULL, NULL,
                &first_invalid_escape);
119             -
120             -       if (v != NULL && first_invalid_escape != NULL) {
121             -           if (warn_invalid_escape_sequence(parser, *first_invalid_escape, t) < 0)
                {
122             -               /* We have not decref u before because first_invalid_escape points
                    117 +       int first_invalid_escape_char;
                    118 +       const char *first_invalid_escape_ptr;
                    119 +       v = _PyUnicode_DecodeUnicodeEscapeInternal2(s, (Py_ssize_t)len, NULL, NULL,
                    120 +                                                   &first_invalid_escape_char,
                    121 +                                                   &first_invalid_escape_ptr);
                    122 +
                    123 +       if (v != NULL && first_invalid_escape_ptr != NULL) {
                    124 +           if (warn_invalid_escape_sequence(parser, *first_invalid_escape_ptr, t)
                < 0) {
```

```
                 +             /* We have not decref u before because first_invalid_escape_ptr points
123  126                          inside u. */
124  127                  Py_XDECREF(u);
125  128                  Py_DECREF(v);
     ...  @@ -133,14 +136,17 @@ decode_unicode_with_escapes(Parser *parser, const char *s, size_t len, Token *t)
133  136    static PyObject *
134  137    decode_bytes_with_escapes(Parser *p, const char *s, Py_ssize_t len, Token *t)
135  138    {
136       -      const char *first_invalid_escape;
137       -      PyObject *result = _PyBytes_DecodeEscape(s, len, NULL,
                 &first_invalid_escape);
     139 +      int first_invalid_escape_char;
     140 +      const char *first_invalid_escape_ptr;
     141 +      PyObject *result = _PyBytes_DecodeEscape2(s, len, NULL,
     142 +                                                &first_invalid_escape_char,
     143 +                                                &first_invalid_escape_ptr);
138  144        if (result == NULL) {
139  145            return NULL;
140  146        }
141  147
142       -      if (first_invalid_escape != NULL) {
143       -          if (warn_invalid_escape_sequence(p, *first_invalid_escape, t) < 0) {
     148 +      if (first_invalid_escape_ptr != NULL) {
     149 +          if (warn_invalid_escape_sequence(p, *first_invalid_escape_ptr, t) < 0)
                 {
144  150                Py_DECREF(result);
145  151                return NULL;
146  152            }
     ...
```

Comments 0