python / **cpython** Public

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

⚠ This commit does not belong to any branch on this repository, and may belong to a fork outside of the repository.

Commit `8d35fd1`

serhiy-storchaka authored on Jun 2, 2025 · ✕ 12 / 15 · Verified

```
[3.9] gh-133767: Fix use-after-free in the unicode-escape decoder with an error
handler (GH-129648) (GH-133944) (#134346)

* [3.9] 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 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)
(cherry picked from commit 8b528ca)

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

· v3.9.25 ••• 3.9

1 parent d4df3c5 commit 8d35fd1

8 files changed +164 -41

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/pegen
  - parse_string.c

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

```
@@ -866,6 +866,19 @@ PyAPI_FUNC(PyObject*)
_PyUnicode_DecodeUnicodeEscapeStateful(
866  866  );
```

```
867  867    /* Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
868  868       chars. */
     869  + PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
     870  +     const char *string,     /* Unicode-Escape encoded string */
     871  +     Py_ssize_t length,      /* size of string */
     872  +     const char *errors,     /* error handling */
     873  +     Py_ssize_t *consumed,   /* bytes consumed */
     874  +     int *first_invalid_escape_char, /* on return, if not -1, contain the first
     875  +                                        invalid escaped char (<= 0xff) or
          invalid
     876  +                                        octal escape (> 0xff) in string. */
     877  +     const char **first_invalid_escape_ptr); /* on return, if not NULL, may
     878  +                                         point to the first invalid escaped
     879  +                                         char in string.
     880  +                                         May be NULL if errors is not NULL. */
     881  + // Export for binary compatibility.
869  882    PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
870  883            const char *string,     /* Unicode-Escape encoded string */
871  884            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_mutatingdecodehandler(self):
     1127  +     def test_mutating_decode_handler(self):
1128 1128              baddata = [
1129 1129                  ("ascii", b"\xff"),
1130 1130                  ("utf-7", b"++"),
@@ -1159,6 +1159,40 @@ 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,
      len(input)))
1181 +             self.assertIn(msg, str(cm.warning))
1182 +
1183 +         check(br'\x0n\z', '\u0404\n\\z', r"invalid escape sequence '\z'")
1184 +         check(br'\x0z', '\u0404\\z', r"invalid escape sequence '\z'")
1185 +
1186 +         check(br'\x3n\zr', '\u0404\n\\zr', r"invalid escape sequence '\z'")
1187 +         check(br'\x3zr', '\u0404\\zr', r"invalid escape sequence '\z'")
1188 +         check(br'\x3z5', '\u0404\\z5', r"invalid escape sequence '\z'")
1189 +         check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r"invalid escape
      sequence '\z'")
1190 +         check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r"invalid escape
      sequence '\z'")
1191 +
1192 +         check(br'\x5n\z', '\u0404\n\\z', r"invalid escape sequence '\z'")
1193 +         check(br'\x5z', '\u0404\\z', r"invalid escape sequence '\z'")
1194 +         check(memoryview(br'\x5zy')[:-1], '\u0404\\z', r"invalid escape
      sequence '\z'")
1195 +
1162 1196         # issue32583
1163 1197         def test_crashing_decode_handler(self):
1164 1198             # better generating one more character to fill the extra space slot
```

Lib/test/test_codecs.py

```
@@ -1178,20 +1178,32 @@ def test_escape(self):
1178 1178            check(br"[\501]", b"[A]")
1179 1179            check(br"[\x41]", b"[A]")
1180 1180            check(br"[\x410]", b"[A0]")
     1181 +
     1182 +    def test_warnings(self):
     1183 +        decode = codecs.escape_decode
     1184 +        check = coding_checker(self, decode)
1181 1185            for i in range(97, 123):
1182 1186                b = bytes([i])
1183 1187                if b not in b'abfnrtvx':
1184      -                 with self.assertWarns(DeprecationWarning):
     1188 +                 with self.assertWarnsRegex(DeprecationWarning,
     1189 +                                            r"invalid escape sequence '\\%c'" % i):
1185 1190                      check(b"\\" + b, b"\\" + b)
1186      -             with self.assertWarns(DeprecationWarning):
     1191 +             with self.assertWarnsRegex(DeprecationWarning,
     1192 +                                        r"invalid escape sequence '\\%c'" % (i-32)):
1187 1193                  check(b"\\" + b.upper(), b"\\" + b.upper())
1188      -         with self.assertWarns(DeprecationWarning):
     1194 +         with self.assertWarnsRegex(DeprecationWarning,
     1195 +                                    r"invalid escape sequence '\\8'"):
1189 1196              check(br"\8", b"\\8")
1190 1197          with self.assertWarns(DeprecationWarning):
1191 1198              check(br"\9", b"\\9")
1192      -         with self.assertWarns(DeprecationWarning):
     1199 +         with self.assertWarnsRegex(DeprecationWarning,
     1200 +                                    r"invalid escape sequence '\\xfa'") as cm:
1193 1201              check(b"\\xfa", b"\\xfa")
1194 1202
     1203 +         with self.assertWarnsRegex(DeprecationWarning,
     1204 +                                    r"invalid escape sequence '\\z'"):
     1205 +             self.assertEqual(decode(br'\x\z', 'ignore'), (b'\\z', 4))
     1206 +
1195 1207      def test_errors(self):
1196 1208          decode = codecs.escape_decode
1197 1209          self.assertRaises(ValueError, decode, br"\x")
```

```
@@ -2393,20 +2405,31 @@ def test_escape_decode(self):
2393 2405             check(br"[\x410]", "[A0]")
2394 2406             check(br"\u20ac", "\u20ac")
2395 2407             check(br"\U0001d120", "\U0001d120")
     2408 +
     2409 +     def test_decode_warnings(self):
     2410 +         decode = codecs.unicode_escape_decode
     2411 +         check = coding_checker(self, decode)
2396 2412             for i in range(97, 123):
2397 2413                 b = bytes([i])
2398 2414                 if b not in b'abfnrtuvx':
2399      -                   with self.assertWarns(DeprecationWarning):
     2415 +                   with self.assertWarnsRegex(DeprecationWarning,
     2416 +                           r"invalid escape sequence '\\%c'" % i):
2400 2417                         check(b"\\" + b, "\\" + chr(i))
2401 2418                 if b.upper() not in b'UN':
2402      -                   with self.assertWarns(DeprecationWarning):
     2419 +                   with self.assertWarnsRegex(DeprecationWarning,
     2420 +                           r"invalid escape sequence '\\%c'" % (i-32)):
2403 2421                         check(b"\\" + b.upper(), "\\" + chr(i-32))
2404      -           with self.assertWarns(DeprecationWarning):
     2422 +           with self.assertWarnsRegex(DeprecationWarning,
     2423 +                   r"invalid escape sequence '\\8'"):
2405 2424                 check(br"\8", "\\8")
2406 2425             with self.assertWarns(DeprecationWarning):
2407 2426                 check(br"\9", "\\9")
2408      -           with self.assertWarns(DeprecationWarning):
     2427 +           with self.assertWarnsRegex(DeprecationWarning,
     2428 +                   r"invalid escape sequence '\\\xfa'") as cm:
2409 2429                 check(b"\\\xfa", "\\\xfa")
     2430 +           with self.assertWarnsRegex(DeprecationWarning,
     2431 +                   r"invalid escape sequence '\\z'"):
     2432 +               self.assertEqual(decode(br'\x\z', 'ignore'), ('\\z', 4))
2410 2433
2411 2434         def test_decode_errors(self):
2412 2435             decode = codecs.unicode_escape_decode
```

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

```
@@ -0,0 +1,2 @@
```

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

Objects/bytesobject.c

```
@@ -1060,10 +1060,11 @@ _PyBytes_FormatEx(const char *format, Py_ssize_t format_len,
 }

 /* Unescape a backslash-escaped string. */
-PyObject *_PyBytes_DecodeEscape(const char *s,
+PyObject *_PyBytes_DecodeEscape2(const char *s,
                                 Py_ssize_t len,
                                 const char *errors,
-                                const char **first_invalid_escape)
+                                int *first_invalid_escape_char,
+                                const char **first_invalid_escape_ptr)
 {
     int c;
     char *p;
@@ -1077,7 +1078,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
         return NULL;
     writer.overallocate = 1;

-    *first_invalid_escape = NULL;
+    *first_invalid_escape_char = -1;
+    *first_invalid_escape_ptr = NULL;

     end = s + len;
     while (s < end) {
@@ -1152,9 +1154,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
             break;

         default:
-            if (*first_invalid_escape == NULL) {
-                *first_invalid_escape = s-1; /* Back up one char, since we've
-                                                already incremented s. */
+            if (*first_invalid_escape_char == -1) {
+                *first_invalid_escape_char = (unsigned char)s[-1];
+                /* Back up one char, since we've already incremented s. */
```

```
                1160 +                 *first_invalid_escape_ptr = s - 1;
1158            1161                   }
1159            1162                   *p++ = '\\';
1160            1163                   s--;
                     @@ -1168,21 +1171,36 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1168            1171       return NULL;
1169            1172   }
1170            1173
                1174 + // Export for binary compatibility.
                1175 + PyObject *_PyBytes_DecodeEscape(const char *s,
                1176 +                                Py_ssize_t len,
                1177 +                                const char *errors,
                1178 +                                const char **first_invalid_escape)
                1179 + {
                1180 +     int first_invalid_escape_char;
                1181 +     return _PyBytes_DecodeEscape2(
                1182 +             s, len, errors,
                1183 +             &first_invalid_escape_char,
                1184 +             first_invalid_escape);
                1185 + }
                1186 +
1171            1187   PyObject *PyBytes_DecodeEscape(const char *s,
1172            1188                                  Py_ssize_t len,
1173            1189                                  const char *errors,
1174            1190                                  Py_ssize_t Py_UNUSED(unicode),
1175            1191                                  const char *Py_UNUSED(recode_encoding))
1176            1192   {
1177                 -     const char* first_invalid_escape;
1178                 -     PyObject *result = _PyBytes_DecodeEscape(s, len, errors,
1179                 -                                              &first_invalid_escape);
                1193 +     int first_invalid_escape_char;
                1194 +     const char *first_invalid_escape_ptr;
                1195 +     PyObject *result = _PyBytes_DecodeEscape2(s, len, errors,
                1196 +                                               &first_invalid_escape_char,
                1197 +                                               &first_invalid_escape_ptr);
1180            1198       if (result == NULL)
1181            1199           return NULL;
1182                 -     if (first_invalid_escape != NULL) {
                1200 +     if (first_invalid_escape_char != -1) {
1183            1201           if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
```

```
1184  1202                                  "invalid escape sequence '\\%c'",
1185        -                               (unsigned char)*first_invalid_escape) < 0) {
      1203  +                               first_invalid_escape_char) < 0) {
1186  1204                  Py_DECREF(result);
1187  1205                  return NULL;
1188  1206              }
```

Objects/unicodeobject.c

```
      @@ -6278,20 +6278,23 @@ PyUnicode_AsUTF16String(PyObject *unicode)
6278  6278  static _PyUnicode_Name_CAPI *ucnhash_CAPI = NULL;
6279  6279
6280  6280  PyObject *
6281        - _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
      6281  + _PyUnicode_DecodeUnicodeEscapeInternal2(const char *s,
6282  6282                                           Py_ssize_t size,
6283  6283                                           const char *errors,
6284  6284                                           Py_ssize_t *consumed,
6285        -                                          const char **first_invalid_escape)
      6285  +                                          int *first_invalid_escape_char,
      6286  +                                          const char **first_invalid_escape_ptr)
6286  6287  {
6287  6288      const char *starts = s;
      6289  +    const char *initial_starts = starts;
6288  6290      _PyUnicodeWriter writer;
6289  6291      const char *end;
6290  6292      PyObject *errorHandler = NULL;
6291  6293      PyObject *exc = NULL;
6292  6294
6293  6295      // so we can remember if we've seen an invalid escape char or not
6294        -    *first_invalid_escape = NULL;
      6296  +    *first_invalid_escape_char = -1;
      6297  +    *first_invalid_escape_ptr = NULL;
6295  6298
6296  6299      if (size == 0) {
6297  6300          if (consumed) {
      @@ -6474,9 +6477,12 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
      *s,
6474  6477              goto error;
6475  6478
```

```
6476  6479            default:
6477        -             if (*first_invalid_escape == NULL) {
6478        -                 *first_invalid_escape = s-1; /* Back up one char, since we've
6479        -                                              already incremented s. */
      6480  +             if (*first_invalid_escape_char == -1) {
      6481  +                 *first_invalid_escape_char = c;
      6482  +                 if (starts == initial_starts) {
      6483  +                     /* Back up one char, since we've already incremented s.
            */
      6484  +                     *first_invalid_escape_ptr = s - 1;
      6485  +                 }
6480  6486              }
6481  6487              WRITE_ASCII_CHAR('\\');
6482  6488              WRITE_CHAR(c);
          @@ -6515,22 +6521,39 @@ _PyUnicode_DecodeUnicodeEscapeInternal(const char
          *s,
6515  6521      return NULL;
6516  6522  }
6517  6523
      6524  + // Export for binary compatibility.
      6525  + PyObject *
      6526  + _PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
      6527  +                                        Py_ssize_t size,
      6528  +                                        const char *errors,
      6529  +                                        Py_ssize_t *consumed,
      6530  +                                        const char **first_invalid_escape)
      6531  + {
      6532  +     int first_invalid_escape_char;
      6533  +     return _PyUnicode_DecodeUnicodeEscapeInternal2(
      6534  +             s, size, errors, consumed,
      6535  +             &first_invalid_escape_char,
      6536  +             first_invalid_escape);
      6537  + }
      6538  +
6518  6539  PyObject *
6519  6540  _PyUnicode_DecodeUnicodeEscapeStateful(const char *s,
6520  6541                                         Py_ssize_t size,
6521  6542                                         const char *errors,
6522  6543                                         Py_ssize_t *consumed)
6523  6544  {
```

```
6524        -     const char *first_invalid_escape;
6525        -     PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal(s, size,
                  errors,
            6545  +     int first_invalid_escape_char;
            6546  +     const char *first_invalid_escape_ptr;
            6547  +     PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal2(s, size,
                  errors,
6526        6548                                                    consumed,
6527        -                                                       &first_invalid_escape);
            6549  +
                  &first_invalid_escape_char,
            6550  +
                  &first_invalid_escape_ptr);
6528        6551          if (result == NULL)
6529        6552              return NULL;
6530        -     if (first_invalid_escape != NULL) {
            6553  +     if (first_invalid_escape_char != -1) {
6531        6554              if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
6532        6555                                   "invalid escape sequence '\\%c'",
6533        -                                  (unsigned char)*first_invalid_escape) < 0) {
            6556  +                                  first_invalid_escape_char) < 0) {
6534        6557                  Py_DECREF(result);
6535        6558                  return NULL;
6536        6559              }
```

Parser/pegen/parse_string.c

```
            @@ -119,12 +119,15 @@ decode_unicode_with_escapes(Parser *parser, const char
            *s, size_t len, Token *t)
119         119           len = p - buf;
120         120           s = buf;
121         121
122         -     const char *first_invalid_escape;
123         -     v = _PyUnicode_DecodeUnicodeEscapeInternal(s, len, NULL, NULL,
                  &first_invalid_escape);
124         -
125         -     if (v != NULL && first_invalid_escape != NULL) {
126         -         if (warn_invalid_escape_sequence(parser, *first_invalid_escape, t) < 0)
                  {
127         -             /* We have not decref u before because first_invalid_escape points
```

```
                 +     int first_invalid_escape_char;
                 +     const char *first_invalid_escape_ptr;
                 +     v = _PyUnicode_DecodeUnicodeEscapeInternal2(s, (Py_ssize_t)len, NULL, NULL,
                 +                                                 &first_invalid_escape_char,
                 +                                                 &first_invalid_escape_ptr);
                 +
                 +     if (v != NULL && first_invalid_escape_ptr != NULL) {
                 +         if (warn_invalid_escape_sequence(parser, *first_invalid_escape_ptr, t)
                   < 0) {
                 +             /* We have not decref u before because first_invalid_escape_ptr
                   points
 128  131                    inside u. */
 129  132                Py_XDECREF(u);
 130  133                Py_DECREF(v);
                   @@ -138,14 +141,17 @@ decode_unicode_with_escapes(Parser *parser, const char
                   *s, size_t len, Token *t)
 138  141    static PyObject *
 139  142    decode_bytes_with_escapes(Parser *p, const char *s, Py_ssize_t len, Token *t)
 140  143    {
 141       -     const char *first_invalid_escape;
 142       -     PyObject *result = _PyBytes_DecodeEscape(s, len, NULL,
                   &first_invalid_escape);
      144  +     int first_invalid_escape_char;
      145  +     const char *first_invalid_escape_ptr;
      146  +     PyObject *result = _PyBytes_DecodeEscape2(s, len, NULL,
      147  +                                               &first_invalid_escape_char,
      148  +                                               &first_invalid_escape_ptr);
 143  149        if (result == NULL) {
 144  150            return NULL;
 145  151        }
 146  152
 147       -     if (first_invalid_escape != NULL) {
 148       -         if (warn_invalid_escape_sequence(p, *first_invalid_escape, t) < 0) {
      153  +     if (first_invalid_escape_ptr != NULL) {
      154  +         if (warn_invalid_escape_sequence(p, *first_invalid_escape_ptr, t) < 0)
                   {
 149  155                Py_DECREF(result);
 150  156                return NULL;
 151  157            }
```

Comments 0