python / **cpython** Public

Code Issues 5k+ Pull requests 2.1k 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 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().
```

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

1 parent `734e15b` commit `9f69a58`

8 files changed +160 -63

Top

Filter files…

- 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

Search within code

Include/internal/pycore_bytesobject.h

```
@@ -20,8 +20,9 @@ extern PyObject* _PyBytes_FromHex(
 
 // Helper for PyBytes_DecodeEscape that detects invalid escape chars.
 // Export for test_peg_generator.
-PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape(const char *, Py_ssize_t,
-                                            const char *, const char **);
+PyAPI_FUNC(PyObject*) _PyBytes_DecodeEscape2(const char *, Py_ssize_t,
+                                             const char *,
+                                             int *, const char **);
 
 
 // Substring Search.
```

Include/internal/pycore_unicodeobject.h

```
@@ -139,14 +139,18 @@ extern PyObject*
 _PyUnicode_DecodeUnicodeEscapeStateful(
 // Helper for PyUnicode_DecodeUnicodeEscape that detects invalid escape
 // chars.
 // Export for test_peg_generator.
-PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal(
+PyAPI_FUNC(PyObject*) _PyUnicode_DecodeUnicodeEscapeInternal2(
     const char *string,     /* Unicode-Escape encoded string */
     Py_ssize_t length,      /* size of string */
     const char *errors,     /* error handling */
     Py_ssize_t *consumed,   /* bytes consumed */
-    const char **first_invalid_escape); /* on return, points to first
-                                           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,
     len(input)))
1183 +             self.assertIn(msg, str(cm.warning))
1184 +
1185 +         check(br'\x0n\z', '\u0404\n\\z', r'"\z" is an invalid escape
     sequence')
1186 +         check(br'\x0n\501', '\u0404\n\u0141', r'"\501" is an invalid octal
     escape sequence')
1187 +         check(br'\x0z', '\u0404\\z', r'"\z" is an invalid escape sequence')
1188 +
1189 +         check(br'\x3n\zr', '\u0404\n\\zr', r'"\z" is an invalid escape
     sequence')
1190 +         check(br'\x3zr', '\u0404\\zr', r'"\z" is an invalid escape sequence')
1191 +         check(br'\x3z5', '\u0404\\z5', r'"\z" is an invalid escape sequence')
1192 +         check(memoryview(br'\x3z5x')[:-1], '\u0404\\z5', r'"\z" is an invalid
     escape sequence')
1193 +         check(memoryview(br'\x3z5xy')[:-2], '\u0404\\z5', r'"\z" is an
     invalid escape sequence')
1194 +
1195 +         check(br'\x5n\z', '\u0404\n\\z', r'"\z" is an invalid escape
     sequence')
```

```
        1196 +         check(br'\x5n\501', '\u0404\n\u0141', r'"\501" is an invalid octal
               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
               escape sequence')
        1199 +
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        -             with self.assertWarns(DeprecationWarning):
      1221  +             with self.assertWarnsRegex(DeprecationWarning,
      1222  +                     r'"\\%o" is an invalid octal escape sequence' % i):
1214  1223                  check(rb'\%o' % i, bytes([i & 0o377]))
1215  1224
      1225  +     with self.assertWarnsRegex(DeprecationWarning,
      1226  +             r'"\\z" is an invalid escape sequence'):
      1227  +         self.assertEqual(decode(br'\x\z', 'ignore'), (b'\\z', 4))
      1228  +     with self.assertWarnsRegex(DeprecationWarning,
      1229  +             r'"\\501" is an invalid octal escape sequence'):
      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'"\\%c" is an invalid escape sequence' % 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'"\\%c" is an invalid escape sequence' % (i-32)):
2671  2693                      check(b"\\" + b.upper(), "\\" + chr(i-32))
2672        -         with self.assertWarns(DeprecationWarning):
      2694  +         with self.assertWarnsRegex(DeprecationWarning,
```

```
        2695 +                 r'"\\8" is an invalid escape sequence'):
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'"\\\xfa" is an invalid escape sequence') 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'"\\%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        -                                 const char **first_invalid_escape)
      1081  +                                 int *first_invalid_escape_char,
      1082  +                                 const char **first_invalid_escape_ptr)
1082  1083    {
1083  1084        int c;
1084  1085        char *p;
      @@ -1092,7 +1093,8 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1092  1093            return NULL;
1093  1094        writer.overallocate = 1;
1094  1095
1095        -     *first_invalid_escape = NULL;
      1096  +     *first_invalid_escape_char = -1;
      1097  +     *first_invalid_escape_ptr = NULL;
1096  1098
1097  1099        end = s + len;
1098  1100        while (s < end) {
      @@ -1130,9 +1132,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1130  1132                        c = (c<<3) + *s++ - '0';
1131  1133                }
1132  1134                if (c > 0377) {
1133        -                   if (*first_invalid_escape == NULL) {
1134        -                       *first_invalid_escape = s-3; /* Back up 3 chars, since
      1135  +                   if (*first_invalid_escape_char == -1) {
      1136  +                       *first_invalid_escape_char = c;
      1137  +                       /* Back up 3 chars, since we've already incremented s. */
      1138  +                       *first_invalid_escape_ptr = s - 3;
1136  1139                        }
1137  1140                }
1138  1141                *p++ = c;
      @@ -1173,9 +1176,10 @@ PyObject *_PyBytes_DecodeEscape(const char *s,
1173  1176                break;
1174  1177
1175  1178            default:
1176        -               if (*first_invalid_escape == NULL) {
```

```
1177        -                *first_invalid_escape = s-1; /* Back up one char, since we've
1178        -                                             already incremented s. */
        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\"\\%o\" 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,
                if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
                                     "b\"\\%c\" is an invalid escape sequence. "
                                     "Such sequences will not work in the future.
",
-                                    c) < 0)
+                                    first_invalid_escape_char) < 0)
                {
                    Py_DECREF(result);
                    return NULL;
```

Objects/unicodeobject.c

```
@@ -6596,21 +6596,24 @@ _PyUnicode_GetNameCAPI(void)
/* --- Unicode Escape Codec ----------------------------------------------
*/

PyObject *
-_PyUnicode_DecodeUnicodeEscapeInternal(const char *s,
+_PyUnicode_DecodeUnicodeEscapeInternal2(const char *s,
                                        Py_ssize_t size,
                                        const char *errors,
                                        Py_ssize_t *consumed,
-                                       const char **first_invalid_escape)
+                                       int *first_invalid_escape_char,
+                                       const char **first_invalid_escape_ptr)
{
    const char *starts = s;
+    const char *initial_starts = starts;
    _PyUnicodeWriter writer;
    const char *end;
    PyObject *errorHandler = NULL;
    PyObject *exc = NULL;
    _PyUnicode_Name_CAPI *ucnhash_capi;

    // so we can remember if we've seen an invalid escape char or not
-    *first_invalid_escape = NULL;
+    *first_invalid_escape_char = -1;
+    *first_invalid_escape_ptr = NULL;
```

```
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,
                                    const char *errors,
                                    Py_ssize_t *consumed)
 {
-    const char *first_invalid_escape;
-    PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal(s, size, errors,
+    int first_invalid_escape_char;
+    const char *first_invalid_escape_ptr;
+    PyObject *result = _PyUnicode_DecodeUnicodeEscapeInternal2(s, size, errors,
                                                               consumed,
-                                                              &first_invalid_escape);
+
+                                                              &first_invalid_escape_char,
+
+                                                              &first_invalid_escape_ptr);
     if (result == NULL)
         return NULL;
-    if (first_invalid_escape != NULL) {
-        unsigned char c = *first_invalid_escape;
-        if ('4' <= c && c <= '7') {
+    if (first_invalid_escape_char != -1) {
+        if (first_invalid_escape_char > 0xff) {
             if (PyErr_WarnFormat(PyExc_DeprecationWarning, 1,
-                                 "\"\\%.3s\" is an invalid octal escape sequence. "
+                                 "\"\\%o\" is an invalid octal escape sequence. "
                                  "Such sequences will not work in the future. ",
-                                 first_invalid_escape) < 0)
+                                 first_invalid_escape_char) < 0)
             {
                 Py_DECREF(result);
                 return NULL;
@@ -6864,7 +6874,7 @@ _PyUnicode_DecodeUnicodeEscapeStateful(const char *s,
             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
207       -                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
      210 +                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,
     &first_invalid_escape);
     223  +    int first_invalid_escape_char;
     224  +    const char *first_invalid_escape_ptr;
     225  +    PyObject *result = _PyBytes_DecodeEscape2(s, len, NULL,
     226  +                                              &first_invalid_escape_char,
     227  +                                              &first_invalid_escape_ptr);
222  228      if (result == NULL) {
223  229          return NULL;
224  230      }
225  231
226       -    if (first_invalid_escape != NULL) {
227       -        if (warn_invalid_escape_sequence(p, s, first_invalid_escape, t) < 0) {
     232  +    if (first_invalid_escape_ptr != NULL) {
     233  +        if (warn_invalid_escape_sequence(p, s, first_invalid_escape_ptr, t) <
     0) {
228  234              Py_DECREF(result);
229  235              return NULL;
230  236          }
```

Comments 0