» Issue30657



This issue tracker **has been migrated to GitHub**, and is currently **read-only**.
For more information, **see the GitHub FAQs in the Python's Developer Guide.**



This issue has been migrated to GitHub:
https://github.com/python/cpython/issues/74842

classification

Title:	[security] CVE-2017-1000158: Unsafe arithmetic in PyString_DecodeEscape		
Type:	security	**Stage**:	resolved
Components:	Interpreter Core	**Versions**:	Python 3.4, Python 3.5, Python 2.7

process

Status:	closed	**Resolution**:	fixed
Dependencies:		**Superseder**:	
Assigned To:	serhiy.storchaka	**Nosy List**:	jaybosamiya, larry, leosilva, ned.deily, serhiy.storchaka, vstinner
Priority:	normal	**Keywords**:	easy (C), patch

Created on **2017-06-13 15:35** by **jaybosamiya**, last changed **2022-04-11 14:58** by **admin**. This issue is now **closed**.

Files

File name	Uploaded	Description	Edit
poc-gen.py	jaybosamiya, 2017-06-13 15:35	Generates poc.py	

Pull Requests

URL	Status	Linked	Edit
PR 2174	merged	jaybosamiya, 2017-06-13 20:14	
PR 4664	merged	hroncok, 2017-12-01 17:59	
PR 4758	merged	hroncok, 2017-12-08 20:33	

Messages (17)

msg295930 - (view) **Author: Jay Bosamiya (jaybosamiya) *** **Date: 2017-06-13 15:35**

```
In Python 2.7, there is a possible integer overflow in
PyString_DecodeEscape function of the file stringobject.c, which can
be abused to gain a heap overflow, possibly leading to arbitrary code
execution.
```

The relevant parts of the code are highlighted below:

```
    PyObject *PyString_DecodeEscape(const char *s,
                                    Py_ssize_t len,
                                    const char *errors,
                                    Py_ssize_t unicode,
                                    const char *recode_encoding)
    {
        int c;
        char *p, *buf;
        const char *end;
        PyObject *v;
(1)     Py_ssize_t newlen = recode_encoding ? 4*len:len;
(2)     v = PyString_FromStringAndSize((char *)NULL, newlen);
        if (v == NULL)
            return NULL;
(3)     p = buf = PyString_AsString(v);
        end = s + len;
        while (s < end) {
            if (*s != '\\') {
              non_esc:
    #ifdef Py_USING_UNICODE
    [...]
    #else
                *p++ = *s++;
    #endif
                continue;
    [...]
            }
        }
(4)     if (p-buf < newlen)
            _PyString_Resize(&v, p - buf); /* v is cleared on error */
        return v;
      failed:
        Py_DECREF(v);
        return NULL;
    }


(1) If recode_encoding is true (i.e., non-null), we have an integer
      overflow here which can set newlen to be some very small value
(2) This allows a small string to be created into v
(3) Now p (and buf) use that small string
(4) The small string is copied into with a larger string, thereby
      giving a heap buffer overflow

In the highly unlikely but definitely possible situation that we pass
it a very large string (in the order of ~1GB on a 32-bit Python
install), one can reliably get heap corruption. It is possible to
access this function (and condition in line(1)) through function
parsestr from ast.c, when the file encoding of an input .py file is
something apart from utf-8 and iso-8859-1. This can be trivially done
using the following at the start of the file:
    # -*- coding: us-ascii -*-
```

The attached file (poc-gen.py) produces a poc.py file which satisfies these constraints and shows the vulnerability.

Note: To see the vulnerability in action, it is necessary to have an ASAN build of Python, compiled for 32 bit on a 64 bit machine. Additionally, the poc.py file generated can take an extremely long time to load (over a few hours), and finally crash. Instead, if one wishes to see the proof of vulnerability quicker, then it might be better to change the constant 4 in line (1) to 65536 (just for simplicity sake), and change the multiplication_constant in poc-gen.py file to be the same (i.e. 65536).

Proposed fix: Confirm that the multiplication will not overflow, before actually performing the multiplication and depending on the result.

msg295947 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-06-13 18:05**

Thank you for your report Jay. Even if it very unlikely that this can occurred unintentionally or be used for attack, this still is a bug and should be fixed. Do you want to provide a patch?

msg295957 - (view) **Author: Jay Bosamiya (jaybosamiya) *** **Date: 2017-06-13 20:19**

I've made a patch that should fix the vulnerability. Please do let me know if changes are required. Thanks a lot :)

PS: For anyone who looks at this later on, in my original message describing the issue, the line `*p++ = *s++;` should be marked as (4) instead to understand this issue better.

msg296277 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-06-18 16:41**

New changeset **c3c9db89273fabc62ea1b48389d9a3000c1c03ae** by Serhiy Storchaka (Jay Bosamiya) in branch '2.7':
[2.7] ~~**bpo-30657**~~: Check & prevent integer overflow in PyString_DecodeEscape (~~**#2174**~~)
https://github.com/python/cpython/commit/c3c9db89273fabc62ea1b48389d9a3000c1c03ae

msg306823 - (view) **Author: Leonidas S. Barbosa (leosilva)** **Date: 2017-11-23 15:11**

I couldn't reproduce using the poc, but seems python3.5 is also vulnerable to this bug. The code from py3.5 are quite similar to 2.7.
In py3.5: **Objects/bytesobject.c** PyBytes_DecodeEscape

msg306826 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-11-23 16:06**

Right, but it is not easy to exploit this bug. You need to parse Python sources longer than 512 MiB in 32-bit Python.

Python 3.5 currently takes only fixes for security bugs. I left on to Larry to decide whether it is worth to port the fix to 3.5.

msg306875 - (view) **Author: Larry Hastings (larry) *** **Date: 2017-11-24 01:34**

I would welcome a backport of this for 3.5 and even 3.4 (if it's vulnerable, which it probably is).

msg306890 - (view) **Author: Leonidas S. Barbosa (leosilva)** **Date: 2017-11-24 11:21**

Python 3.4 also has the similar code as 3.5, but applying the same patch tests for it results in test errors:

```
+======================================================================
+ERROR: test_anydbm_creation (test.test_dbm.TestCase-dbm.ndbm)
+----------------------------------------------------------------------
+Traceback (most recent call last):
+  File "/<<PKGBUILDDIR>>/Lib/test/test_dbm.py", line 74, in
test_anydbm_creation^M
+    self.read_helper(f)^M
+  File "/<<PKGBUILDDIR>>/Lib/test/test_dbm.py", line 115, in read_helper
+    self.assertEqual(self._dict[key], f[key.encode("ascii")])
+KeyError: b'0'
+
+======================================================================
+ERROR: test_anydbm_modification (test.test_dbm.TestCase-dbm.ndbm)
+----------------------------------------------------------------------
+Traceback (most recent call last):
+  File "/<<PKGBUILDDIR>>/Lib/test/test_dbm.py", line 89, in
test_anydbm_modification
+    self.read_helper(f)
+  File "/<<PKGBUILDDIR>>/Lib/test/test_dbm.py", line 115, in read_helper
+    self.assertEqual(self._dict[key], f[key.encode("ascii")])
+KeyError: b'0'
+
+======================================================================
+ERROR: test_anydbm_read (test.test_dbm.TestCase-dbm.ndbm)
+----------------------------------------------------------------------
+Traceback (most recent call last):
+  File "/<<PKGBUILDDIR>>/Lib/test/test_dbm.py", line 95, in test_anydbm_read
+    self.read_helper(f)
+  File "/<<PKGBUILDDIR>>/Lib/test/test_dbm.py", line 115, in read_helper
+    self.assertEqual(self._dict[key], f[key.encode("ascii")])
+KeyError: b'0'
```

msg307138 - (view) **Author: Leonidas S. Barbosa (leosilva)** **Date: 2017-11-28 15:59**

I re-did the build here for python3.4 and couldn't reach the same test fail. So I'm assuming it was a false alarm.
Said that, I believe the same patch that applies to py2.7 also applies to 3.4 and 3.5. I've build them using the patch and did some regression tests and it was ok.

msg307238 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-11-29 16:05**

Leo kirotawa silva: "I re-did the build here for python3.4 and couldn't reach the same test fail. So I'm assuming it was a false alarm."

Python 3.4 and 3.5 seem to be also vulnerable:

```
PyObject *PyBytes_DecodeEscape(const char *s,
                               Py_ssize_t len,
                               const char *errors,
                               Py_ssize_t unicode,
                               const char *recode_encoding)
{
    ...
    Py_ssize_t newlen = recode_encoding ? 4*len:len;
    v = PyBytes_FromStringAndSize((char *)NULL, newlen);
---

I don't think that Python 3.6 and 3.7 are vulnerable, the code was rewritten with
the _PyBytesWriter API. The code got a new _PyBytes_DecodeEscapeRecode() helper
function which calls _PyBytesWriter_WriteBytes(), and this function detects
properly integer overflows.
```

msg307240 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-11-29 16:21**

I added this vulnerability to the python-security website:
http://python-security.readthedocs.io/vuln/cve-2017-1000158_pystring_decodeescape_integer_overflow.html

msg307243 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-11-29 17:03**

I don't think it is worth to add this vulnerability to the python-security website. You need to compile a 1 GiB Python file on 32-bit system for reproducing it. It is very unlikely that this can happen by accident, and it is hard to used it in security attack. If you can make the attacked program compiling a 1 GiB Python file, you perhaps have easier ways to make a harm.

msg307245 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-11-29 17:09**

Serhiy: "I don't think it is worth to add this vulnerability to the python-security website. You need to compile a 1 GiB Python file on 32-bit system for reproducing it. It is very unlikely that this can happen by accident, and it is hard to used it in security attack. If you can make the attacked program compiling a 1 GiB Python file, you perhaps have easier ways to make a harm."

I'm trying to keep track of all CVEs. People are scared by CVE numbers :-(But it seems like any bug can get a CVE number, without any real evalution of the severity of the bug.

I completed the description on python-security with your paragraph.

FYI I wrote python-security to make sure that vulnerabilities are fixed in supported Python branches. Here it seems like we forgot to fix Python 3.4 and 3.5.

msg307246 - (view) **Author: Larry Hastings (larry) *** **Date: 2017-11-29 17:11**

Resetting to "needs patch", because we still need PRs for 3.4 and 3.5 (please!).

msg307868 - (view) **Author: Larry Hastings (larry) *** **Date: 2017-12-08 21:34**

New changeset **fd8614c5c5466a14a945db5b059c10c0fb8f76d9** by larryhastings (Miro Hrončok) in branch '3.5':
~~**bpo-30657**~~: Fix CVE-2017-1000158 (~~**#4664**~~)
https://github.com/python/cpython/commit/fd8614c5c5466a14a945db5b059c10c0fb8f76d9

msg307869 - (view) **Author: Larry Hastings (larry) *** **Date: 2017-12-08 21:34**

New changeset **6c004b40f9d51872d848981ef1a18bb08c2dfc42** by larryhastings (Miro Hrončok) in branch '3.4':
~~**bpo-30657**~~: Fix CVE-2017-1000158 (~~**#4758**~~)
https://github.com/python/cpython/commit/6c004b40f9d51872d848981ef1a18bb08c2dfc42

msg307870 - (view) **Author: Larry Hastings (larry) *** **Date: 2017-12-08 21:36**

Merged into 3.4 and 3.5. Thanks for the patches!

Since I see 2.7 has already had the fix committed, and these are the only three versions affected, I'm marking as closed / resolved / fixed.

History

Date	User	Action	Args
2022-04-11 14:58:47	admin	set	github: 74842
2019-05-10 18:20:34	ned.deily	set	messages: - **msg342086**
2019-05-10 17:36:37	ned.deily	set	nosy: + **ned.deily** messages: + **msg342086**
2017-12-08 21:36:52	larry	set	status: open -> closed resolution: fixed messages: + **msg307870** stage: patch review -> resolved
2017-12-08 21:34:46	larry	set	messages: + **msg307869**
2017-12-08 21:34:19	larry	set	messages: + **msg307868**
2017-12-08 20:33:27	hroncok	set	pull_requests: + **pull_request4661**
2017-12-01 17:59:39	hroncok	set	keywords: + **patch** stage: needs patch -> patch review pull_requests: + **pull_request4574**
2017-11-29 17:11:48	larry	set	messages: + **msg307246** stage: resolved -> needs patch
2017-11-29 17:09:27	vstinner	set	messages: + **msg307245**
2017-11-29 17:03:20	serhiy.storchaka	set	messages: + **msg307243**
2017-11-29 16:21:22	vstinner	set	messages: + **msg307240**
2017-11-29 16:19:18	vstinner	set	title: CVE-2017-1000158: Unsafe arithmetic in PyString_DecodeEscape -> [security] CVE-2017-1000158: Unsafe arithmetic in PyString_DecodeEscape

2017-11-29 16:05:39	vstinner	set	status: closed -> open title: Unsafe arithmetic in PyString_DecodeEscape -> CVE-2017-1000158: Unsafe arithmetic in PyString_DecodeEscape nosy: + **vstinner** versions: + Python 2.7, Python 3.5 messages: + **msg307238** resolution: fixed -> (no value)
2017-11-28 15:59:51	leosilva	set	messages: + **msg307138**
2017-11-24 11:21:24	leosilva	set	messages: + **msg306890** versions: + Python 3.4, - Python 2.7
2017-11-24 01:34:56	larry	set	messages: + **msg306875**
2017-11-23 16:06:04	serhiy.storchaka	set	nosy: + **larry** messages: + **msg306826**
2017-11-23 15:11:00	leosilva	set	nosy: + **leosilva** messages: + **msg306823**
2017-06-18 16:44:15	serhiy.storchaka	set	status: open -> closed resolution: fixed stage: patch review -> resolved
2017-06-18 16:41:06	serhiy.storchaka	set	messages: + **msg296277**
2017-06-16 09:23:22	serhiy.storchaka	set	assignee: **serhiy.storchaka** stage: needs patch -> patch review
2017-06-13 20:19:19	jaybosamiya	set	messages: + **msg295957**
2017-06-13 20:14:54	jaybosamiya	set	pull_requests: + **pull_request2226**
2017-06-13 18:05:21	serhiy.storchaka	set	keywords: + **easy (C)** messages: + **msg295947** stage: needs patch
2017-06-13 16:01:16	vstinner	set	nosy: + **serhiy.storchaka**
2017-06-13 15:35:29	jaybosamiya	create	