» Issue31530



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/75711

classification

Title: CVE-2018-1000030: Python 2.7 readahead feature of file objects is not thread safe	
Type: security	**Stage**: resolved
Components: Interpreter Core	**Versions**: Python 2.7

process

Status: closed	**Resolution**: fixed
Dependencies:	**Superseder**:
Assigned To:	**Nosy List**: benjamin.peterson, gvanrossum, lemburg, pitrou, serhiy.storchaka, vstinner
Priority: normal	**Keywords**: patch

Created on **2017-09-20 13:27** by **vstinner**, last changed **2022-04-11 14:58** by **admin**. This issue is now **closed**.

Files

File name	Uploaded	Description	Edit
readahead.py	vstinner, 2017-09-20 13:27		
0001-stop-crashes-when-iterating-over-a-file-on-multiple-.patch	vstinner, 2017-09-20 13:28		

Pull Requests

URL	Status	Linked	Edit
PR 3670	closed	benjamin.peterson, 2017-09-20 14:35	
PR 3672	merged	serhiy.storchaka, 2017-09-20 15:31	
PR 5060	merged	benjamin.peterson, 2017-12-31 05:27	

Messages (25)

msg302610 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 13:27**

Reading from the same file object in different threads does crash Python 2.7. The readahead feature of **Objects/fileobject.c** is not thread safe.

Try attached script to reproduce the crash.

msg302611 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 13:28**

Benjamin Peterson proposed attached patch.

@Benjamin: Would you mind to convert this patch to a PR to ease review?

msg302613 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 13:36**

The bug was first reported to the private Python security mailing list. The PSRT decided that it's a regular bug and doesn't need to be categorized as a vulnerability, since the attacker has to be able to run arbitrary code in practice.

The PSRT considers that no Python 2.7 application currently rely on reading from the same file object "at the same time" from different thread, since it currently crashs.

So an attacker would have to run his/her own code... but if an attacker can already run arbitrary code, why relying on an unstable race condition and inject machine code (so not portable), whereas Python standard library is full of nice features to write your portable exploit?

For more information, see the Python security model:
https://python-security.readthedocs.io/security.html#security-model

msg302614 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 13:39**

Python 3 is not affected by this bug.

In Python 3, the full I/O stack was rewritten from scratchn, the new io module has a different design. Reading ahead still exists in the io module, but it is now done by a dedicated object: io.BufferedReader, and this object uses a lock to prevent concurrent reads. A single thread controls the file position at the same time. (Except if a different thread uses directly the file descriptor, but that's a different story.)

msg302615 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 13:53**

In Python 3, reading ahead is implemented by _io.BufferedReader. This object uses a lock to provide a prevent race condition: it's not only to prevent crashes, but also provide warranties on how the file is read.

If thread A calls read() first, it gets the next bytes. If thread B calls read() while thread A is filling the internal file buffer ("readahead buffer"?), the second read is queued. The file position is only controlled by a single thread at the same time.

_PyOS_URandom() uses a similar strategy than Benjamin's proposed patch for the cached file descriptor of /dev/urandom:

```
    fd = _Py_open("/dev/urandom", O_RDONLY);
```

```
    if (fd < 0) {
        ...
        return -1;
    }
    if (urandom_cache.fd >= 0) {
        /* urandom_fd was initialized by another thread while we were
           not holding the GIL, keep it. */
        close(fd);
        fd = urandom_cache.fd;
    }
    else {
        ...
        urandom_cache.fd = fd;
    }
```

The difference is that opening /dev/urandom multiple times in parallel is safe, whereas reading from the same file descriptor in parellel... using the buffered fread()... is not safe. readahead() can require multiple fread() calls, so multiple read() syscalls. Interlaced reads in parallel is likely to return scrambled data.

Adding a lock in Python 2.7.15 can impact performances even on single threaded applications.

I'm not sure what whaters more here: performance or correctness?

Note: Even the awesome Python 3 io module has same flaws!
https://bugs.python.org/issue12215 "TextIOWrapper: issues with interlaced read-write"

The question is more *who* reads from the same file object in parallel? Does it make sense? :-) Do you expect that file.read(n) is "atomic" in term of parallelism?

Note 2: the io module is also available in Python 2.7, just not used by default by the builtin open() function ;-) io.open() must be used explicitly.

msg302616 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-09-20 13:53**

The patch is complex. What if just deny reentrant reads? Set a flag while read into a buffer, check it before reading in other thread, and raise RuntimeError.

msg302618 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 14:04**

Serhiy: "What if just deny reentrant reads? Set a flag while read into a buffer, check it before reading in other thread, and raise RuntimeError."

io.BufferedReader/io.BufferedWriter raises a RuntimeError exception for reentrant call, but only in the same thread. For example, it ease debug for signal handlers which trigger such reentrant call.

I'm not sure about 0001-stop-crashes-when-iterating-over-a-file-on-multiple-.patch since it doesn't fix the consistency: two parallel readline() calls can return the same line, instead of being mutual exclusive and only return different lines.

I'm not sure about adding a new lock. "Lock" sounds like "dead locks". I dislike the risk of introducing dead locks very late in the Python 2.7 development cycle.

I like the idea of a simple exception on concurrent operations. But I'm not sure how much code it will break :-/ Crazy idea: would it make sense to raise an exception by default, but add an opt-in option to ignore the exception? I wrote "crazy" since it became clear that the code is not thread-safe and so that parallel operations on the same file object is likely to corrupt data in various funny ways.

msg302619 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-20 14:05**

@Serhiy: Would you like to propose a PR to implement your RuntimeError. Maybe we can test a few popular Python projects to see if the change would break them?

Which popular applications use threads (and files)? :-)

msg302621 - (view) **Author: Guido van Rossum (gvanrossum) *** **Date: 2017-09-20 14:28**

@benjamin can you post your patch as a PR so you'll get credit for it?

msg302623 - (view) **Author: Marc-Andre Lemburg (lemburg) *** **Date: 2017-09-20 14:38**

Why not simply document the fact that read ahead in Python 2.7
is not thread-safe and leave it at that ?

.next() and .readline() already don't work well together, so this
would just add one more case.

msg302625 - (view) **Author: Marc-Andre Lemburg (lemburg) *** **Date: 2017-09-20 14:40**

Ah, didn't see Benjamin's patch: much better solution :-)

msg302626 - (view) **Author: Guido van Rossum (gvanrossum) *** **Date: 2017-09-20 15:22**

> Why not simply document the fact that read ahead in Python 2.7
> is not thread-safe and leave it at that ?

Program bugs should not crash the interpreter. (ctypes excepted.)

msg302627 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-09-20 15:33**

Actually such flag already exists. It is unlocked_count.

There is also similar issue with seek().

msg302631 - (view) **Author: Marc-Andre Lemburg (lemburg) *** **Date: 2017-09-20 15:59**

On 20.09.2017 17:22, Guido van Rossum wrote:
>
>> Why not simply document the fact that read ahead in Python 2.7
>> is not thread-safe and leave it at that ?
>

> Program bugs should not crash the interpreter. (ctypes excepted.)

Ideally not, agreed :-)

msg302932 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-25 09:14**

It's difficult to make a choice between Benjamin's ~~PR 3670~~ and Serhiy's ~~PR 3672~~.

Benjamin wrote that about his PR:

> No attempt is made to define or provide "reasonable" semantics for iterating
> over a file on multiple threads. (Non-crashing) races are still
> present. Duplicated, corrupt, and missing data will happen.

I'm not confortable with the idea of data corruption. I'm not confortable with breaking backward compatibility neither. IMHO the key question is if anyone actually use a file object at the same time in multiple threads. Since, it's likely that iterating on the same object crash, I'm not sure that anyone do it.

If someone rely on the feature, we should maybe explain how to handle properly the issue: use a lock.

I have a preference for Serhiy's ~~PR 3672~~. It's simple, the error is easy to understand. Maybe his PR just lacks a mention in "Porting to Python 2.7" section of What's New in Python 2.7 and/or the NEWS entry. Suggest to use a lock for example.

msg302938 - (view) **Author: Antoine Pitrou (pitrou) *** **Date: 2017-09-25 09:38**

I'm wary of raising an exception. Assuming Benjamin's patch is correct, I vote for it.

msg302939 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-25 09:39**

Antoine: "I'm wary of raising an exception. Assuming Benjamin's patch is correct, I vote for it."

Can you please elaborate? You are fine with the fact that iter(file) in two thread can return the same line?

msg302940 - (view) **Author: Antoine Pitrou (pitrou) *** **Date: 2017-09-25 09:45**

Le 25/09/2017 à 11:39, STINNER Victor a écrit :
>
> Antoine: "I'm wary of raising an exception. Assuming Benjamin's patch is correct, I vote for it."
>
> Can you please elaborate? You are fine with the fact that iter(file) in two thread can return the same line?

AFAIU it already could. Iteration on Python 2 files is fragile, now is not the time to fix fundamental bugs in Python 2.

msg302973 - (view) **Author: Guido van Rossum (gvanrossum) *** **Date: 2017-09-25 19:14**

It should be like 'if key in dct: del dct[key]'. It may fail when two
threads do this but the *internal* state of dct should not be compromised.

On Sep 25, 2017 2:45 AM, "Antoine Pitrou" <**report@bugs.python.org**> wrote:

>
> Antoine Pitrou added the comment:
>
> Le 25/09/2017 à 11:39, STINNER Victor a écrit :
> >
> > Antoine: "I'm wary of raising an exception. Assuming Benjamin's patch
> is correct, I vote for it."
> >
> > Can you please elaborate? You are fine with the fact that iter(file) in
> two thread can return the same line?
>
> AFAIU it already could. Iteration on Python 2 files is fragile, now is
> not the time to fix fundamental bugs in Python 2.
>
> ----------
>
> _____________________________________
> Python tracker <**report@bugs.python.org**>
> <**https://bugs.python.org/issue31530**>
> _____________________________________
>

msg302977 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2017-09-25 19:52**

> Iteration on Python 2 files is fragile, now is not the time to fix fundamental
bugs in Python 2.

Right. I don't have a strong preference for Serhiy's ~~**PR 3672**~~.

msg306021 - (view) **Author: Serhiy Storchaka (serhiy.storchaka) *** **Date: 2017-11-10 10:58**

New changeset **6401e5671781eb217ee1afb4603cc0d1b0367ae6** by Serhiy Storchaka in
branch '2.7':
[2.7] ~~**bpo-31530**~~: Stop crashes when iterating over a file on multiple threads.
(~~**#3672**~~)
https://github.com/python/cpython/commit/6401e5671781eb217ee1afb4603cc0d1b0367ae6

msg309265 - (view) **Author: Benjamin Peterson (benjamin.peterson) *** **Date: 2017-12-31 05:24**

Unfortunately, it looks like this fix causes a regression. Some programs rely on
being able to seek() and write to a file on multiple threads. For example,
py.test captures the standard streams by redirecting them to a tmpfile and then
truncating+seeking to 0 after every test. This change broke that situation when
multiple threads are logging. Anyway, there's no real fundamental reason to
prevent concurrent access, since the underlying stdio implementation is
threadsafe. I think we'll have to resurrect my PR.

msg309386 - (view) **Author: Benjamin Peterson (benjamin.peterson) *** **Date: 2018-01-02 17:25**

New changeset **dbf52e02f18dac6f5f0a64f78932f3dc6efc056b** by Benjamin Peterson in branch '2.7':
~~**bpo-31530**~~: fix crash when multiple threads iterate over a file, round 2 (~~**#5060**~~)
https://github.com/python/cpython/commit/dbf52e02f18dac6f5f0a64f78932f3dc6efc056b

msg320189 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2018-06-21 14:48**

This bug has been fixed in Python 2.7.15.

msg320190 - (view) **Author: STINNER Victor (vstinner) *** **Date: 2018-06-21 15:00**

I added this issue to my python-security page since the issue got a CVE number: CVE-2018-1000030

* **http://python-security.readthedocs.io/vuln/cve-2018-1000030_python_2.7_readahead_is_not_thread_safe.html**
* **https://access.redhat.com/security/cve/cve-2018-1000030**

History

Date	User	Action	Args
2022-04-11 14:58:52	admin	set	github: 75711
2018-06-21 15:00:21	vstinner	set	type: security messages: + **msg320190** title: Python 2.7 readahead feature of file objects is not thread safe -> CVE-2018-1000030: Python 2.7 readahead feature of file objects is not thread safe
2018-06-21 14:48:42	vstinner	set	messages: + **msg320189**
2018-01-02 17:31:30	benjamin.peterson	set	status: open -> closed resolution: fixed stage: patch review -> resolved
2018-01-02 17:25:49	benjamin.peterson	set	messages: + **msg309386**
2017-12-31 05:27:48	benjamin.peterson	set	stage: resolved -> patch review pull_requests: + **pull_request4936**
2017-12-31 05:25:00	benjamin.peterson	set	status: closed -> open resolution: fixed -> (no value) messages: + **msg309265**
2017-11-10 10:59:55	serhiy.storchaka	set	status: open -> closed resolution: fixed stage: patch review -> resolved
2017-11-10 10:58:58	serhiy.storchaka	set	messages: + **msg306021**
2017-09-25 19:52:04	vstinner	set	messages: + **msg302977**
2017-09-25 19:14:43	gvanrossum	set	messages: + **msg302973**
2017-09-25 09:45:03	pitrou	set	messages: + **msg302940**
2017-09-25 09:39:54	vstinner	set	messages: + **msg302939**

2017-09-25 09:38:46	pitrou	set	messages: + **msg302938**
2017-09-25 09:14:35	vstinner	set	messages: + **msg302932**
2017-09-20 15:59:59	lemburg	set	messages: + **msg302631** title: [2.7] Python 2.7 readahead feature of file objects is not thread safe -> Python 2.7 readahead feature of file objects is not thread safe
2017-09-20 15:33:08	serhiy.storchaka	set	messages: + **msg302627**
2017-09-20 15:31:15	serhiy.storchaka	set	pull_requests: + **pull_request3661**
2017-09-20 15:22:24	gvanrossum	set	messages: + **msg302626**
2017-09-20 14:40:54	lemburg	set	messages: + **msg302625**
2017-09-20 14:38:01	lemburg	set	nosy: + **lemburg** messages: + **msg302623**
2017-09-20 14:35:05	benjamin.peterson	set	stage: patch review pull_requests: + **pull_request3660**
2017-09-20 14:28:59	gvanrossum	set	nosy: + **gvanrossum** messages: + **msg302621**
2017-09-20 14:05:29	vstinner	set	messages: + **msg302619**
2017-09-20 14:04:17	vstinner	set	messages: + **msg302618**
2017-09-20 13:53:57	serhiy.storchaka	set	messages: + **msg302616**
2017-09-20 13:53:06	vstinner	set	nosy: + **pitrou** messages: + **msg302615**
2017-09-20 13:39:52	vstinner	set	messages: + **msg302614**
2017-09-20 13:36:42	vstinner	set	messages: + **msg302613**
2017-09-20 13:28:32	vstinner	set	files: + **0001-stop-crashes-when-iterating-over-a-file-on-multiple-.patch** nosy: + **benjamin.peterson**, **serhiy.storchaka** messages: + **msg302611** keywords: + **patch**
2017-09-20 13:27:12	vstinner	create	