zipfile.Path regression #123270

New issue

Open **obfusk** opened this issue last week · 22 comments · Fixed by jaraco/zipp#124

obfusk commented last week • edited by bedevere-app bot Contributor

Bug report

Bug description:

#122906 introduced a regression with directories that look like Windows drive letters (on Linux):

```
>>> import io, zipfile
>>> zf = zipfile.ZipFile(io.BytesIO(), "w")
>>> zf.writestr("d:/foo", "bar")
>>> zf.extractall("a")
>>> open("a/d:/foo").read()
'bar'
>>> p = zipfile.Path(zf)
>>> x = p / "d" / "foo"
>>> y = p / "d:" / "foo"
>>> list(p.iterdir())   # before: [Path(None, 'd:/')
[Path(None, 'd/')]
>>> p.root.namelist()   # before: ['d:/foo', 'd:/']
['d/foo', 'd/']
>>> x.exists()          # before: False
True
>>> y.exists()          # before: True
False
>>> zf.extractall("b")  # before: worked like above
KeyError: "There is no item named 'd/foo' in the arc
>>> x.read_text()       # before: FileNotFoundError
KeyError: "There is no item named 'd/foo' in the arc
>>> y.read_text()       # before: worked
FileNotFoundError: ...
```

This is the result of `_sanitize()` unconditionally treating a directory that looks like a drive letter as such and removing the colon, regardless of operating system:

cpython/Lib/zipfile/_path/__init__.py
Line 141 in 58fdb16

```
141     bare = re.sub('^([A-Z]):', r'\1', name, fla
```

Assignees

jaraco

Labels

3.8 3.9 3.10 3.11 3.12 3.13 3.14 stdlib type-bug type-security

Projects

Zipfile issues

Status: Done

Milestone

No milestone

Development

Successfully merging a pull request may close this issue.

Restore support for special filenames
jaraco/zipp

4 participants

Whereas `_extract_member()` uses `os.path.splitdrive()` (which is a no-op on Linux):

cpython/Lib/zipfile/__init__.py
Line 1807 in `58fdb16`

```
1807        arcname = os.path.splitdrive(arcname)[1]
```

CPython versions tested on:

3.12

Operating systems tested on:

Linux

Linked PRs

- gh-123270: Replaced SanitizedNames with a more surgical fix. #123354
- [3.13] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123410
- [3.12] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123411
- [3.11] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123425
- [3.10] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123426
- [3.9] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123432
- [3.8] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123433

obfusk added the type-bug label last week

obfusk mentioned this issue last week

gh-122905: Sanitize names in zipfile.Path. #122906 Merged

obfusk changed the title ~~zipfile.Path regression with directories that look like Windows drive letters (on Linux)~~ **zipfile.Path regression** last week

obfusk commented last week Contributor Author

Paths like `foo//bar` and `foo\\bar` also cause issues:

```
>>> import io, zipfile
>>> zf = zipfile.ZipFile(io.BytesIO(), "w")
>>> zf.writestr("foo//bar", "text")
>>> zf.namelist()
['foo//bar']
>>> p = zipfile.Path(zf)
>>> p.root.namelist()   # before: ['foo//bar', 'foo/
['foo/bar', 'foo/']
>>> x = p / "foo//bar"
>>> y = p / "foo" / "bar"
>>> x.exists()          # before: True
False
>>> y.exists()          # before: False
True
>>> x.read_text()       # before: works, returns 'te
FileNotFoundError: ...
>>> y.read_text()       # before: FileNotFoundError
KeyError: "There is no item named 'foo/bar' in the a
```

```
>>> import io, zipfile
>>> zf = zipfile.ZipFile(io.BytesIO(), "w")
>>> zf.writestr("foo\\bar", "text")
>>> zf.namelist()
['foo\\bar']
>>> p = zipfile.Path(zf)
>>> p.root.namelist()   # before: ['foo\\bar']
['foo/bar', 'foo/']
>>> x = p / "foo\\bar"
>>> y = p / "foo" / "bar"
>>> x.exists()          # before: True
False
>>> y.exists()          # before: False
True
>>> x.read_text()       # before: works, returns 'te
FileNotFoundError: ...
>>> y.read_text()       # before: FileNotFoundError
KeyError: "There is no item named 'foo/bar' in the a
```

ZeroIntensity commented last week Contributor

cc @jaraco

obfusk commented last week Contributor Author

Sanitising is good. But e.g. `.open()` will not work if it can't map back to the original path to pass that to `ZipFile.open()`. Though `.joinpath()` will of course never use `//` or `\\`, so opening these files already required constructing the path without using that. Not sure what's best here if opening files with paths affected by sanitising is meant to be supported.

picnixz added the stdlib label last week

obfusk commented last week Contributor Author

```
# before
>>> import io, os, zipfile
>>> zf = zipfile.ZipFile(io.BytesIO(), "w")
>>> zf.writestr("foo/bar", "one")
>>> zf.writestr("foo\\bar", "two")
>>> zf.namelist()
['foo/bar', 'foo\\bar']
>>> zf.extractall("a")
>>> os.system("tree a")
a
├── foo
│   └── bar
└── foo\bar
>>> p = zipfile.Path(zf)
>>> p.root.namelist()
['foo/bar', 'foo\\bar', 'foo/']
>>> (p / "foo" / "bar").read_text()
'one'
>>> (p / "foo\\bar").read_text()
'two'

# after
>>> import io, os, zipfile
>>> zf = zipfile.ZipFile(io.BytesIO(), "w")
>>> zf.writestr("foo/bar", "one")
>>> zf.writestr("foo\\bar", "two")
>>> p = zipfile.Path(zf)
>>> zf.namelist()
['foo/bar', 'foo/bar', 'foo/']
>>> zf.extractall("a")
>>> os.system("tree a")
a
└── foo
    └── bar
>>> (p / "foo" / "bar").read_text()
'one'
>>> (p / "foo\\bar").read_text()
FileNotFoundError: ...
```

obfusk commented last week Contributor Author

> Sanitising is good. But e.g. `.open()` will not work if it can't map back to the original path to pass that to `ZipFile.open()`. Though `.joinpath()` will of course never use `//` or `\\`, so opening these files already required constructing the path without using that. Not sure what's best here if opening files with paths affected by sanitising is meant to be supported.

Keeping a `dict` mapping sanitised paths back to the originals might be a good choice. Though `.namelist()` will still be "wrong" for the ZipFile (e.g. when calling `.extractall()`). And it doesn't handle "duplicates" (see above).

jaraco commented last week Member

> Not sure what's best here if opening files with paths affected by sanitising is meant to be supported.

My expectation was that paths affected by sanitizing would not be supported.

> regression with directories that look like Windows drive letters (on Linux)

My goal is to provide a (perhaps constrained) interface that's portable and provides consistent behavior independent of platform.

I'll have to think about this some more

> Paths like `foo//bar` and `foo\\bar` also cause issues:

I was not expecting that `foo\bar` or `d:/foo` were in use in zip files.

Can you say more about the use-cases that are affected by these paths? What causes these zip files to exist? What does the author of such a zip file expect to happen when consumed by a zipfile client?

obfusk commented 5 days ago • edited Contributor Author

> I was not expecting that `foo\bar` or `d:/foo` were in use in zip files.

Are these file names common? Maybe not. But on Linux these are perfectly acceptable file names. And whilst they may indeed cause portability issues for Windows users and I would thus avoid them when creating ZIP files for use on Windows, the ZIP format also considers them perfectly acceptable.

I myself have quite a few music files with odd file names; song titles can be quite varied. And whilst I've sometimes run into problems extracting those files on other operating systems I have never had issues creating or listing ZIP files.

File names like `foo/..//./bar` can be considered malformed and would not be generated by simply creating a ZIP file from an existing directory tree. But `d:/foo\\bar` is not malformed, just unusual (and perhaps not portable).

```
$ mkdir d:
$ echo "Hi!" >> d:/foo\\bar
$ zip -r foo.zip d:
  adding: d:/ (stored 0%)
  adding: d:/foo\bar (stored 0%)
$ unzip -l foo.zip
Archive:  foo.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  2024-08-24 04:04   d:/
        4  2024-08-24 04:04   d:/foo\bar
---------                     -------
        4                     2 files
$ python3 -mzipfile -l foo.zip
File Name
Modified             Size
d:/
2024-08-24 04:04:20            0
d:/foo\bar
2024-08-24 04:04:20            4
$ zipinfo.py -e foo.zip
Archive:  foo.zip
Zip file size: 308 bytes, number of entries: 2
drwx------  3.0 unx        0 bx        0 stor
2024-08-24 04:04:20 00000000 d:/
-rw-------  3.0 unx        4 tx        4 stor
2024-08-24 04:04:20 54a8a39c d:/foo\bar
2 files, 4 bytes uncompressed, 4 bytes
compressed:  0.0%
```

Tools like `zip` and `unzip` and my own `zipinfo.py` handle them just fine. So I would expect the same from Python. And indeed `zipfile.ZipFile` handles them just fine. But `zipfile.Path` now rejects them.

I generally expect to be able to take any directory tree and turn it into a ZIP file without having to worry about file names containing colons or backslashes. Or at the most that I might need to be careful when extracting them on Windows. I would not expect to have problems merely listing the contents, especially when not even using Windows.

So it is very surprising when `zipfile.Path` effectively rejects perfectly valid ZIP files just because they have "unsupported" file names, especially since it didn't do so before. I would perhaps expect this kind of backwards-incompatible change to happen in a major version, given a good reason, but not in a security patch. And for it to be clearly documented if the interface is constrained like this. It also fails in unexpected ways, instead of providing a clear error message that makes it clear the problem is an unsupported file name.

obfusk commented 5 days ago Contributor Author

> My expectation was that paths affected by sanitizing would not be supported.

And my expectation would be that such paths would be either explicitly rejected as malformed if they cannot reasonably be supported. Or when they can be, handled gracefully with a portable interface that maps sanitised names providing a consistent interface that can reasonably be supported onto the underlying file names instead of simply causing attempts to read files to fail because the underlying file name does not match the sanitised one. And before this change, `d:/foo\\bar` worked just fine.

obfusk commented 5 days ago Contributor Author

This worked perfectly fine before. Now it raises `FileNotFoundError`.

```
>>> zf = zipfile.ZipFile("foo.zip", "r")
>>> zf.namelist()
['d:/', 'd:/foo\\bar']
>>> p = zipfile.Path(zf)
>>> (p / "d:").is_dir()
True
>>> (p / "d:" / "foo\\bar").read_text()
'Hi!\n'
```

And if `d:/foo\\bar` is not considered a valid path by `zipfile.Path`, I'd expect an error. Instead it happily tells me there is a file named `d/foo/bar`. Except when I try reading I get an error that says there is no such file:

```
>>> (p / "d" / "foo" / "bar").is_file()
True
>>> (p / "d" / "foo" / "bar").read_text()
KeyError: "There is no item named 'd/foo/bar' in the
```

obfusk commented 5 days ago   Contributor   Author

Note that it's not just `d:/foo` that's no longer allowed, e.g. `V: The New Mythology Suite.flac` is now also a forbidden file name (though `extractall()` would remove the "V: " -- but only on Windows, and only when extracting, not listing files).

Meanwhile, this seems incorrect:

```
>>> zipfile._path.SanitizedNames._sanitize("d:    ')
'd/foo'
>>> zipfile._path.SanitizedNames._sanitize("/d:/foo"
'd:/foo'
```

obfusk commented 5 days ago   Contributor   Author

Meanwhile, as I posted here, the only fix needed for the infinite loop would be the loop condition here:

cpython/Lib/zipfile/_path/__init__.py
Lines 53 to 55 in 52caaef

```
53      while path and path != posixpath.sep:
54          yield path
55          path, tail = posixpath.split(path)
```

This should work:

```
    while path and path != posixpath.sep * len    )
        yield path
        path, tail = posixpath.split(path)
```

Or this:

```
    while path:
        yield path
        head, tail = posixpath.split(path)
        if head == path:
            break
        path = head
```

jaraco commented 3 days ago • edited   Member

It seems I've stumbled into a hornets nest. When I received the report about the infinite loops, my objective was to try to understand the broader problem and not simply address an issue with the infinite loops. I sought to figure out why `zipfile.extractall` was not affected by the same issue, which led me to the sanitizing behavior that it applies. I drew inspiration from that in an attempt to generalize the kinds of unsupported, malformed paths. At the same time, I took the opportunity to be more constraining in order to provide a uniform experience on any platform. As originally designed, `zipfile.Path` aims to support basic, commonly constructed zip files.

> Note that it's not just `d:/foo` that's no longer allowed, e.g. `V: The New Mythology Suite.flac` is now also a forbidden file name (though `extractall()` would remove the "V: " -- but only on Windows, and only when extracting, not listing files).

Yeah, that's unfortunate, but I think zipfile.Path does not wish to support this name because it isn't valid on commonly-found operating systems... or to support the name, but in a uniform way. In my opinion, it's more desirable to have a consistent behavior than to have varying behavior depending on platform.

On the other hand, supporting this name for traversal on any operating system should be fine, right up until the point that these files are manifest on the file system, such as `importlib.resources.as_file` does here.

~~zipfile.Path aims to represent the zipfile in a way that's safe to "extract all contents" on any platform, which means it can't expose `V: The New Mythology...` as any component of the path.~~

> Meanwhile, this seems incorrect:
>
> ```
> >>> zipfile._path.SanitizedNames._sanitize
> 'd/foo'
> >>> zipfile._path.SanitizedNames._sanitize("/d:
> 'd:/foo'
> ```

Yes, I agree. Probably all colons should be disallowed.

> `.open()` will not work if it can't map back to the original path to pass that to `ZipFile.open()`

That's a good point, and not something I considered when sanitizing, and really suggests that sanitizing early is much more complicated than it sounds.

> Meanwhile, as I posted here, the only fix needed for the infinite loop would be the loop condition here:

The more I think about it, the more I'm liking this approach. Leave the path segments unaffected and let them fail if the user uses a name that's not valid on a platform. In that way, the path `V: The New Mythology` would be valid on any platform and would fail with the appropriate error if the user tried to "open" that name on a Windows OS (same for `d:/` or `/d:/`).

The real problem, I think, is that we also want to protect against other separators, like `\` or special names like `..`.

Should zipfile.Path consider `\` as a path separator? Looking at the spec:

> 4.4.17.1 The name of the file, with optional relative path. The path stored MUST NOT contain a drive or device letter, or a leading slash. All slashes MUST be forward slashes '/' as opposed to backwards slashes '' for compatibility with Amiga and UNIX file systems etc.

So it seems reasonable to just treat backslashes and dots and everything else as part of the path segment and not a path separator, and let the OS fail if anything attempts to create those as files or directories in the system (e.g. `mkdir('\\')` or `open('..', 'w')`).

I'm now convinced, `SanitizedNames` should be scrapped and replaced by a surgical fix for the infinite loop in the traversal.

👍 1

jaraco commented 3 days ago Member

In jaraco/zipp#124, I've started work on the issue.

jaraco self-assigned this 3 days ago

jaraco added a commit to jaraco/zipp that referenced this issue 3 days ago

`Address infinite loop when zipfile begins with more than one leading …` ··· Verified ✕ `9bee3b9`

jaraco added a commit to jaraco/zipp that referenced this issue 3 days ago

Address infinite loop when zipfile begins with more than one leading … ··· Verified 4aab5e4

jaraco mentioned this issue 3 days ago

Restore support for special filenames jaraco/zipp#124 Merged

jaraco added a commit to jaraco/zipp that referenced this issue 3 days ago

Address infinite loop when zipfile begins with more than one leading … ··· Verified f89b93f

jaraco commented 3 days ago Member

> This should work:
>
> ```
>     while path and path != posixpath.sep
>         yield path
>         path, tail = posixpath.split(path)
> ```

I tried applying this, and it does address the infinite loop. I ended up using `not path.endswith(posixpath.sep)`

In jaraco/zipp@ `0a3a7b4` , I had to adjust the expectation for `.iterdir()` - it no longer emits any elements that begin with forward slashes. I think that's fine. Silently ignoring malformed paths is better than presenting them under another name and then not being able to open them.

👍 1

obfusk commented 3 days ago Contributor Author

> So it seems reasonable to just treat backslashes and dots and everything else as part of the path segment and not a path separator, and let the OS fail if anything attempts to create those as files or directories in the system (e.g. `mkdir('\\')` or `open('..', 'w')`).

Using `ZipFile.extractall()` is safe because it sanitises the paths *when extracting*. AFAIK `zipfile.Path` does not provide its own interface to extract files, but any code that traverses the Path and replicates it to disk (like the `importlib` code you linked) would need to be careful to not be vulnerable to path traversal and perhaps try to gracefully handle filenames not supported by the operating system (and that would apply to any Path, not just ZIP files).

> I tried applying this, and it does address the infinite loop. I ended up using `not path.endswith(posixpath.sep)`

One of several essentially equivalent fixes (assuming no change in behaviour of `posixpath.split()`). Maybe even the cleanest (I hadn't really decided which I preferred yet) :)

jaraco commented 3 days ago Member

> One of several essentially equivalent fixes (assuming no change in behaviour of `posixpath.split()`). Maybe even the cleanest (I hadn't really decided which I preferred yet) :)

I found something I think I like even better:

```diff
diff --git a/zipp/__init__.py b/zipp/__init__.
index 0b7b44325fe..a3f0b1b481f 100644
--- a/zipp/__init__.py
+++ b/zipp/__init__.py
@@ -65,7 +65,7 @@ def _ancestry(path):
         ['//b//d///f', '//b//d', '//b']
         """
         path = path.rstrip(posixpath.sep)
-        while path and not path.endswith(posixpath.sep)
+        while path.rstrip(posixpath.sep):
             yield path
             path, tail = posixpath.split(path)
```

Because it combines the check for "empty" and "is only slashes" into one check.

👍 2

ZeroIntensity commented 3 days ago Contributor

Is the plan to revert #122906 from all the security branches, and just put the fix for this on >3.11? Or is this going to be backported into all the security-only versions as well?

jaraco commented 3 days ago Member

My plan is to apply the change to security-only versions. Reverting the change would revive the vulnerability.

jaraco commented 3 days ago • edited Member

> AFAIK `zipfile.Path` does not provide its own interface to extract files, but any code that traverses the Path and replicates it to disk (like the `importlib` code you linked) would need to be careful to not be vulnerable to path traversal and perhaps try to gracefully handle filenames not supported by the operating system (and that would apply to any Path, not just ZIP files).

Thanks for raising this concern. I was thinking about it too, and here's my thinking:

Because of the way that `zipfile.Path` represents a tree and not a list of paths, it doesn't require sanitization. Thinking about the `importlib` code, it will start at the root (or some subpath of the zipfile), and name by name construct the dirs and files, so it will invoke `mkdir("d:")` or `open(r"Back\Path", "w")` or `open(r"..", "w")`. Those operations will succeed on platforms where that's allowed and fail where not allowed, but it never (?) has the unintended consequence of causing a user to unexpectedly traverse to a parent directory or a root folder, because each path element is encountered in order and must succeed on `mkdir` or `write_bytes` before traversing further. I'm confident but not certain that covers all of the bases.

obfusk commented 3 days ago Contributor Author

I agree that *the specific code* from `importlib` is likely to be fine -- assuming `mkdir()` fails for `..` etc. on all platforms and we didn't overlook anything else. But slightly different code -- e.g. code that traverses multiple trees that may overlap and thus uses `child.mkdir(exist_ok=True)` -- can easily become vulnerable to path traversal.

jaraco commented 3 days ago Member

> `child.mkdir(exist_ok=True)`

Yeah, that does appear potentially problemmatic. I see a few possible options:

- Revive something like `SanitizedNames` and have the `zipfile.Path` object only expose sanitized names (even for `.open` and `.read*` operations). Possibly make the sanitization platform-sensitive (yuck).
- Advertise that `zipfile.Path` does not do any name sanitization and it's the responsibility of the caller to check the inputs, etc.
- Emit a warning when encountering potentially unsafe paths. Maybe configure the warning be an error by default but allow it to be tuned down to emit an error message or be silenced.
- zipfile.Path could provide its own traversal that could offer some safety checks.

To be sure, the draft as proposed does still address the originally-reported vulnerability, so it's not a regression in security from that perspective, but still it would be nice not to open up prospects for other vulnerabilities.

jaraco closed this as completed in jaraco/zipp#124 3 days ago

---

jaraco reopened this 3 days ago

jaraco added type-security 3.11 3.10 3.9 3.8 3.12 3.13 3.14 labels 3 days ago

jaraco added a commit to jaraco/cpython that referenced this issue 3 days ago

`pythongh-123270: Replaced SanitizedNames with a more surgical fix.` … Verified ✗ `943a462`

jaraco added a commit to jaraco/cpython that referenced this issue 3 days ago

`pythongh-123270: Replaced SanitizedNames with a more surgical fix.` … Verified ✓ 55c2795

bedevere-app bot mentioned this issue 3 days ago

gh-123270: Replaced SanitizedNames with a more surgical fix. #123354 Merged

jaraco commented 3 days ago Member

The fix has been released in zipp 3.20.1. Please feel free to try that out and make sure it addresses your use-case before it lands in CPython.

github-actions bot pushed a commit to aio-libs/aiohttp that referenced this issue 2 days ago

`Bump zipp from 3.20.0 to 3.20.1 (#8914)` … Verified ✓ eb77a4c

jaraco added a commit that referenced this issue 2 days ago

`gh-123270: Replaced SanitizedNames with a more surgical fix. (#123354)` … Verified ✗ 2231286

miss-islington pushed a commit to miss-islington/cpython that referenced this issue 2 days ago

`pythongh-123270: Replaced SanitizedNames with a more surgical fix. (p…` … ✓ 88e9785

miss-islington pushed a commit to miss-islington/cpython that referenced this issue 2 days ago

`pythongh-123270: Replaced SanitizedNames with a more surgical fix. (p…` … ✓ a2ee3af

This was referenced 2 days ago

[3.13] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123410 Open

[3.12] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123411 Open

github-actions bot pushed a commit to aio-libs/aiohttp that referenced this issue yesterday

`Bump zipp from 3.20.0 to 3.20.1 (#8923)` … Verified b0a97da

jaraco added a commit to jaraco/cpython that referenced this issue yesterday

`[3.11] pythongh-123270: Replaced SanitizedNames with a more surgical …` … Verified ✓ 17b77bb

bedevere-app bot mentioned this issue yesterday

[3.11] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123425 Open

jaraco added a commit to jaraco/cpython that referenced this issue yesterday

`[3.10] [3.11] pythongh-123270: Replaced SanitizedNames with a more su…` … Verified ✓ c94488e

bedevere-app bot mentioned this issue yesterday

[3.10] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123426 Open

jaraco added a commit to jaraco/cpython that referenced this issue yesterday

`[3.9] [3.11] pythongh-123270: Replaced SanitizedNames with a more sur…` … Verified 66d3383

bedevere-app bot mentioned this issue yesterday

[3.9] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123432 Open

jaraco added a commit to jaraco/cpython that referenced this issue yesterday

`[3.8] [3.9] [3.11] pythongh-123270: Replaced SanitizedNames with a mo…` Verified `dcb320a`

…

bedevere-app bot mentioned this issue yesterday

[3.8] gh-123270: Replaced SanitizedNames with a more surgical fix. (GH-123354) #123433 Open