Why Windows can’t follow WSL symlinks

blog.trailofbits.com/2024/02/12/why-windows-cant-follow-wsl-symlinks/

By Yarden Shafir

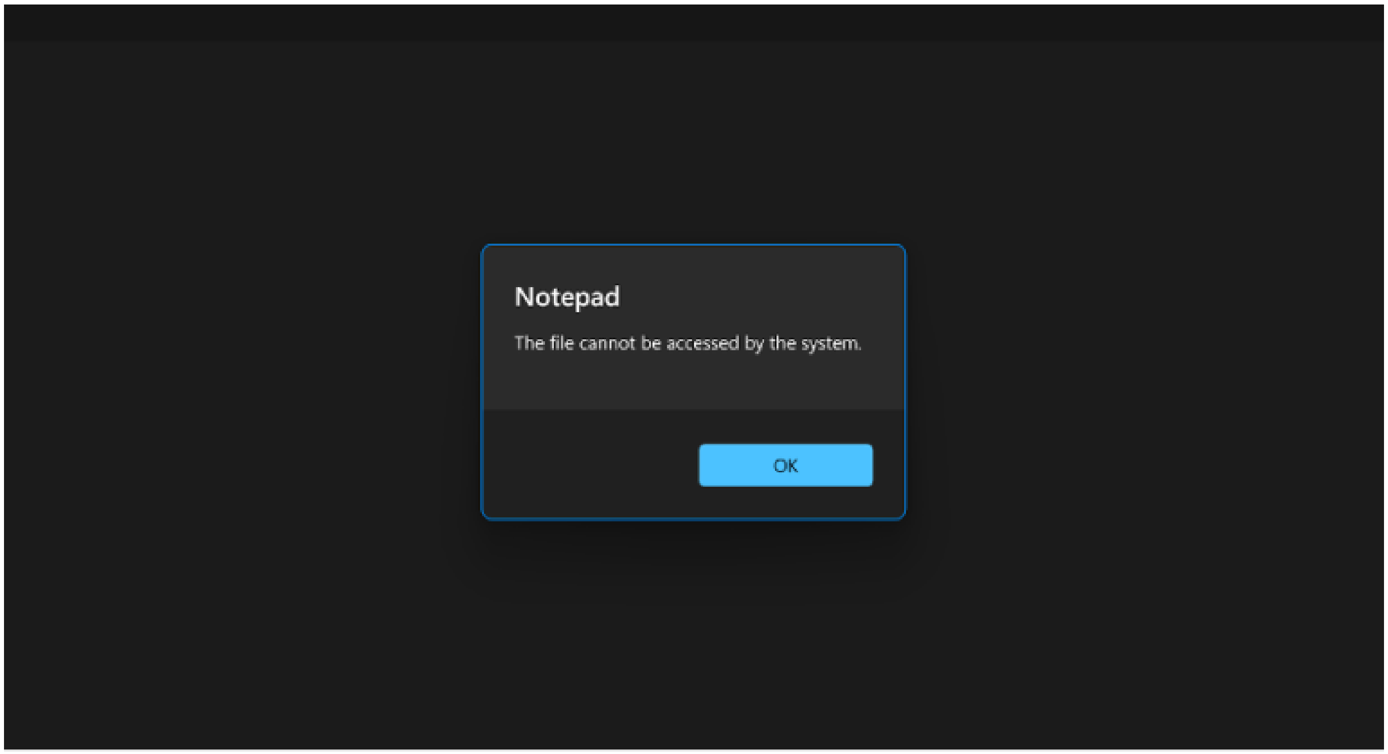
Did you know that symbolic links (or symlinks) created through Windows Subsystem for Linux (WSL) can’t be followed by Windows?

I recently encountered this rather frustrating issue as I’ve been using WSL for my everyday work over the last few months. No doubt others have noticed it as well, so I wanted to document it for anyone who may be seeking answers.

Let’s look at an example of the issue. I’ll use Ubuntu as my Linux client with WSL2 and create a file followed by a symlink to a file in the same directory (via `ln -s`):

```
echo "this is a symlink test" > test_symlink.txt
ln -s test_symlink.txt targetfile.txt
```

In WSL, I can easily read both the original file (`test_symlink.txt`) and the symlink (`targetfile.txt`). But when I try to open the symlink from the Windows file explorer, an error occurs:

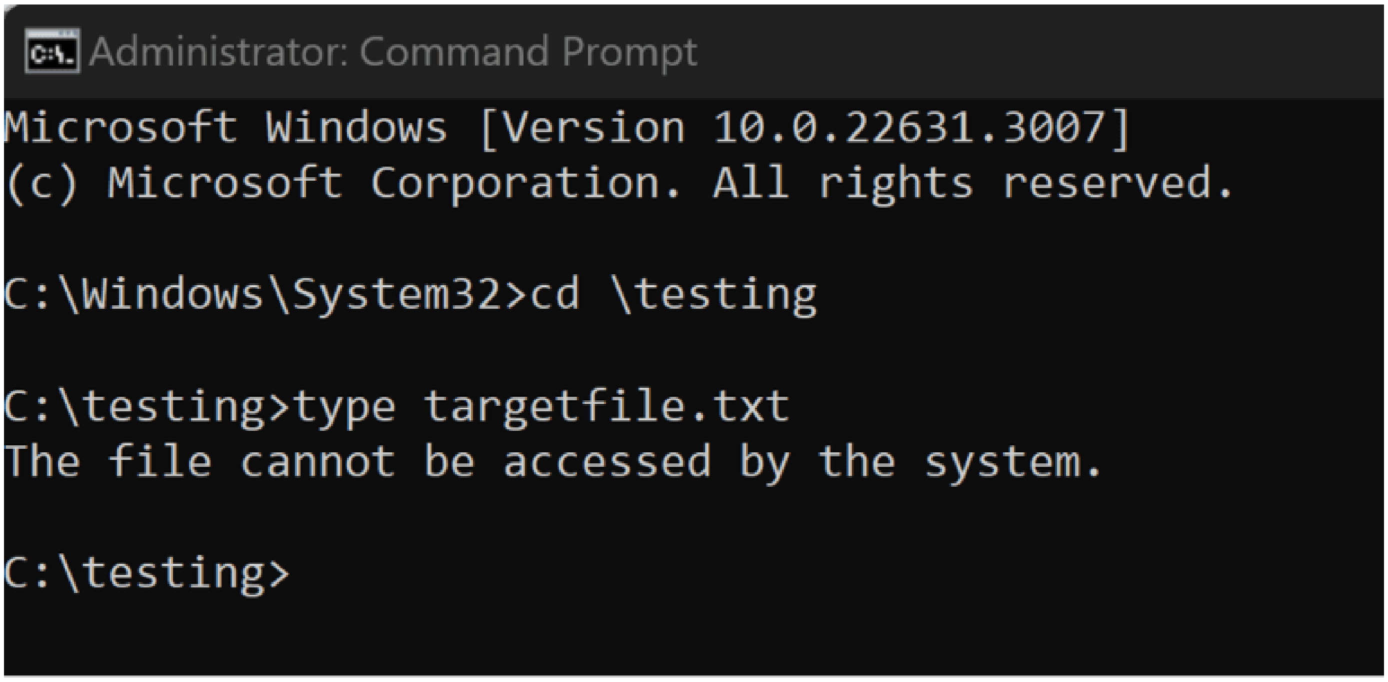


Notepad

The file cannot be accessed by the system.

The Windows file explorer error

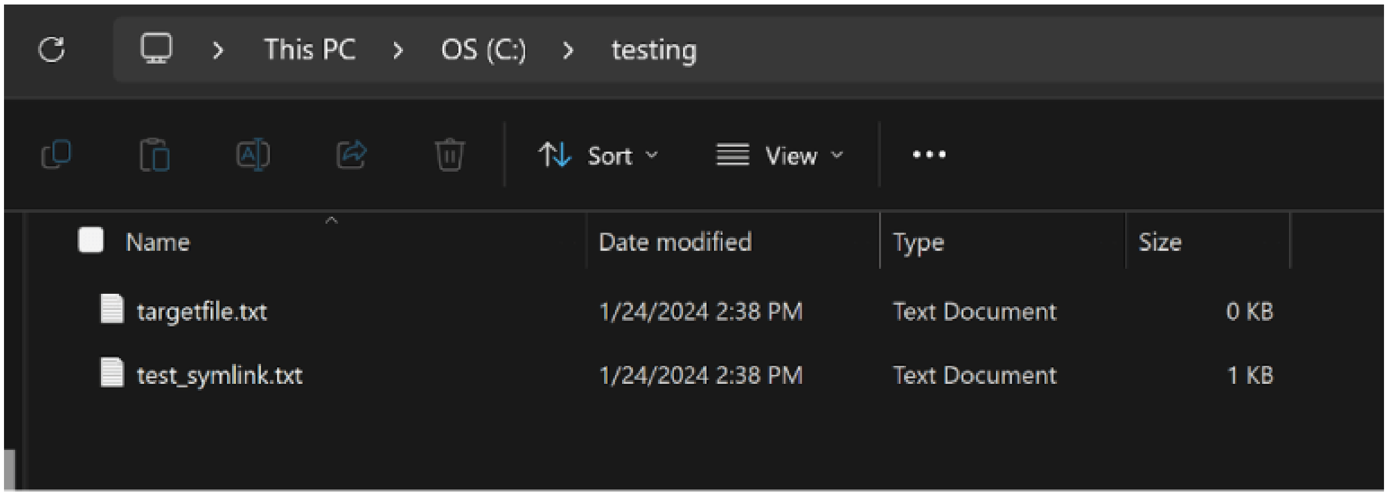
The same error occurs when I try to access `targetfile.txt` from the command line:



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.3007]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd \testing

C:\testing>type targetfile.txt
The file cannot be accessed by the system.

C:\testing>
```

The command line error

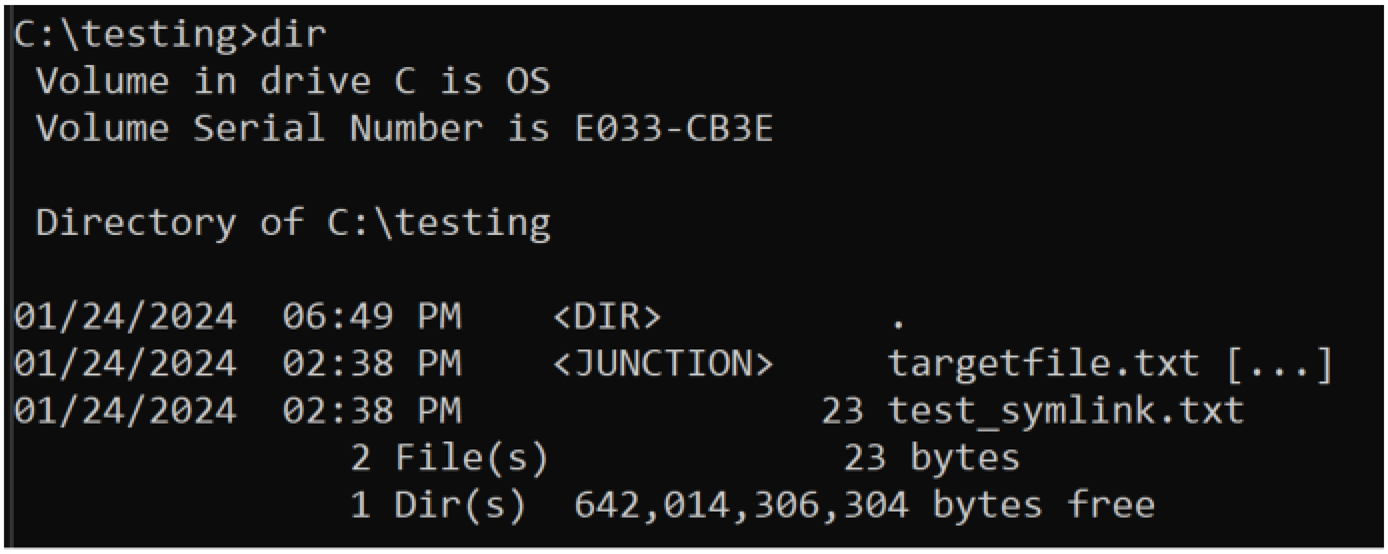
Looking at the directory, I can see the target file, but it has a size of 0 KB:



Name	Date modified	Type	Size
targetfile.txt	1/24/2024 2:38 PM	Text Document	0 KB
test_symlink.txt	1/24/2024 2:38 PM	Text Document	1 KB

The symlink in the directory with a size of 0 KB

And when I run `dir`, I can see that Windows recognizes `targetfile.txt` as an NTFS junction but can't find where the link points to, like it would for a native Windows symlink:

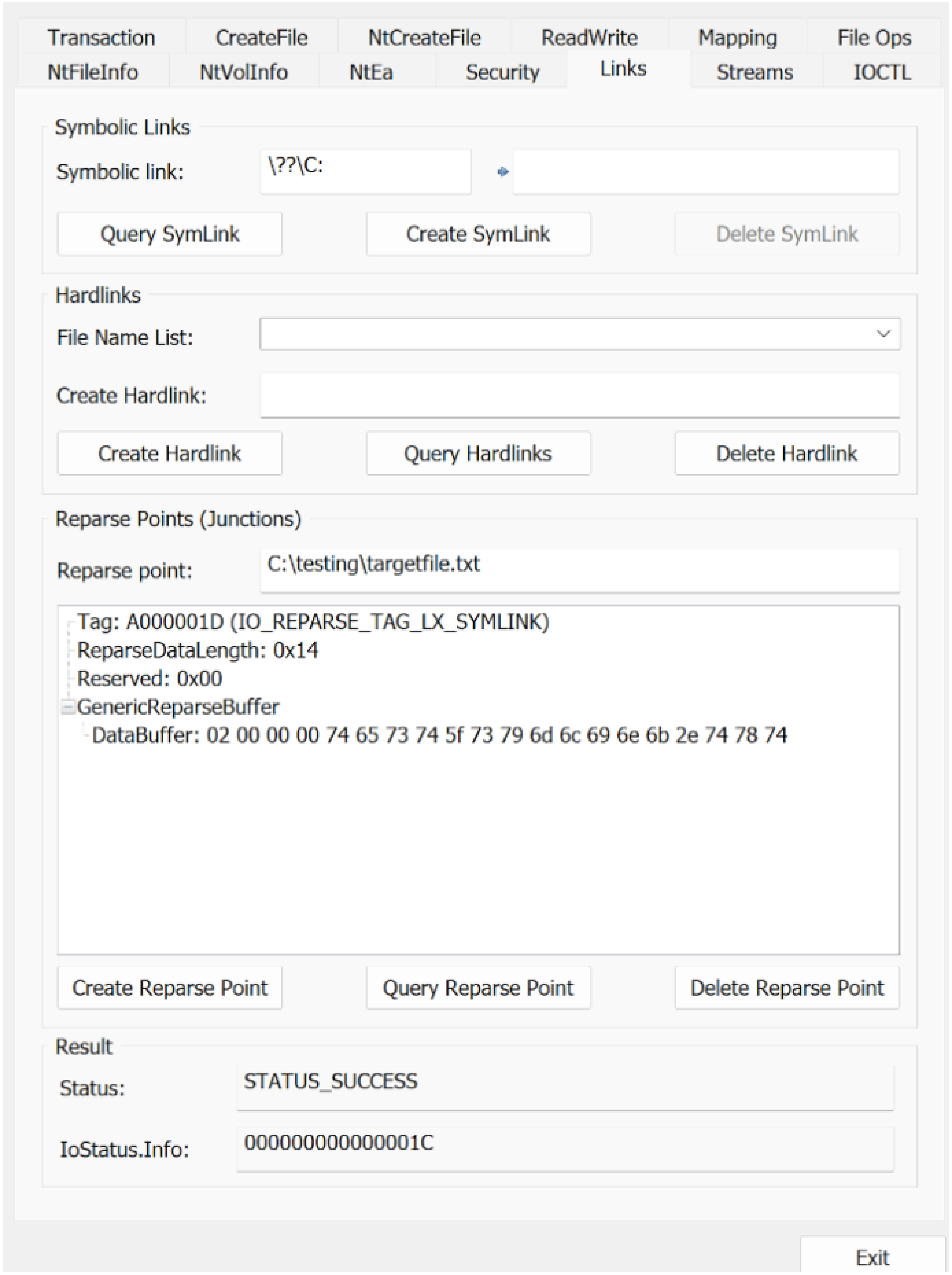


```
C:\testing>dir
 Volume in drive C is OS
 Volume Serial Number is E033-CB3E

 Directory of C:\testing

01/24/2024  06:49 PM    <DIR>          .
01/24/2024  02:38 PM    <JUNCTION>     targetfile.txt [...]
01/24/2024  02:38 PM                23 test_symlink.txt
               2 File(s)             23 bytes
               1 Dir(s)  642,014,306,304 bytes free
```

Windows can't find where the link points to.

When I asked about this behavior on Twitter, Bill Demirkapi had an answer—the link that is created by WSL is an “LX symlink,” which isn't recognized by Windows. That's because symlinks on Linux are implemented differently than symlinks on Windows: on Windows, a symlink is an object, implemented and interpreted by the kernel. On Linux, a symlink is simply a file with a special flag, whose content is a path to the destination. The path doesn't even have to be valid!

Using FileTest, we can easily verify that this is a Linux symlink, not a Windows link. If you look carefully, you can even see the path to the destination file in the file's DataBuffer:



Symbolic link: \??\C:

File Name List:

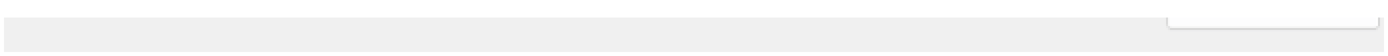
Create Hardlink:

Reparse point: C:\testing\targetfile.txt

```
Tag: A000001D (IO_REPARSE_TAG_LX_SYMLINK)
ReparseDataLength: 0x14
Reserved: 0x00
GenericReparseBuffer
  DataBuffer: 02 00 00 00 74 65 73 74 5f 73 79 6d 6c 69 6e 6b 2e 74 78 74
```

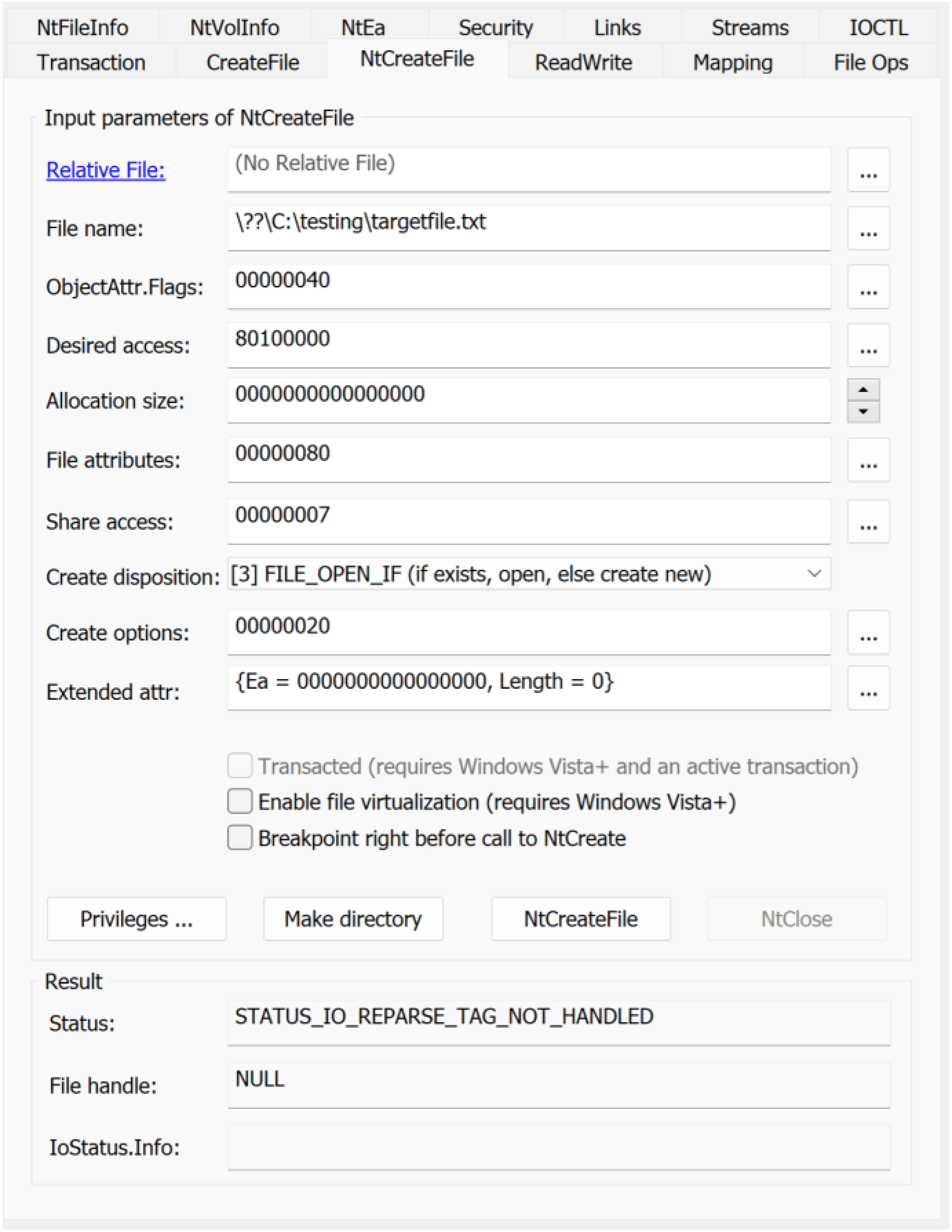
Status: STATUS_SUCCESS

IoStatus.Info: 000000000000001C



FileTest verifies the link as a Linux symlink.

FileTest can also provide a more specific error message regarding the file open failure:



Input parameters of NtCreateFile

Relative File:	(No Relative File)
File name:	\??\C:\testing\targetfile.txt
ObjectAttr.Flags:	00000040
Desired access:	80100000
Allocation size:	0000000000000000
File attributes:	00000080
Share access:	00000007
Create disposition:	[3] FILE_OPEN_IF (if exists, open, else create new)
Create options:	00000020
Extended attr:	{Ea = 0000000000000000, Length = 0}

☐ Transacted (requires Windows Vista+ and an active transaction)
☐ Enable file virtualization (requires Windows Vista+)
☐ Breakpoint right before call to NtCreate

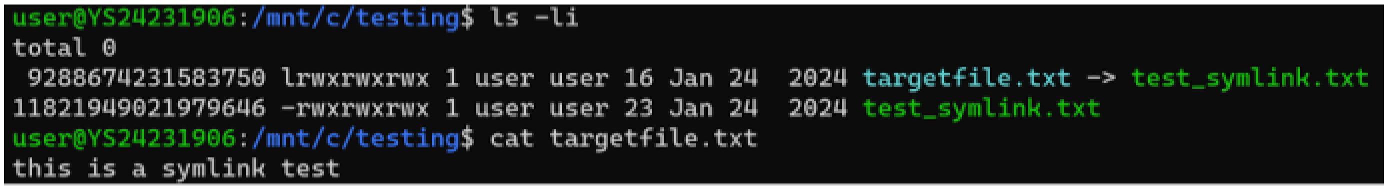
Result

Status:	STATUS_IO_REPARSE_TAG_NOT_HANDLED
File handle:	NULL
IoStatus.Info:	

FileTest's file open failure error message

It turns out that trying to open this file with `NtCreateFile` fails with an `STATUS_IO_REPARSE_TAG_NOT_HANDLED` error, meaning that Windows recognizes this file as a reparse point but can't identify the LX symlink tag and can't follow it. Windows knows how to handle some parts of the Linux filesystem, as explained by Microsoft, but that doesn't include the Linux symlink format.

If I go back to WSL, the symlink works just fine—the system can see the symlink target and open the file as expected:



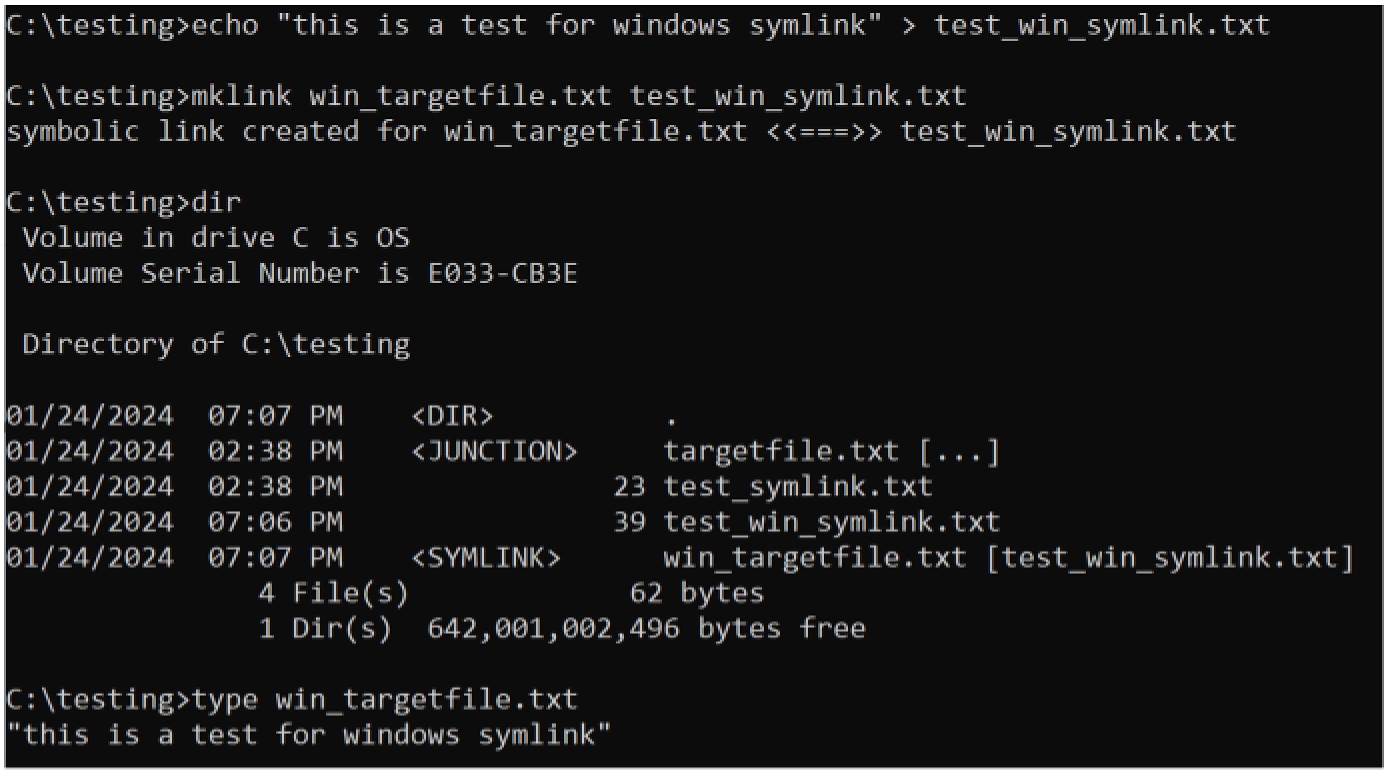
```
user@YS24231906:/mnt/c/testing$ ls -li
total 0
 9288674231583750 lrwxrwxrwx 1 user user 16 Jan 24  2024 targetfile.txt -> test_symlink.txt
11821949021979646 -rwxrwxrwx 1 user user 23 Jan 24  2024 test_symlink.txt
user@YS24231906:/mnt/c/testing$ cat targetfile.txt
this is a symlink test
```

The symlink works in WSL.

It's interesting to note that symlinks created on Windows work normally on WSL. I can create a new file in the same directory and create a symlink for it using the Windows command line (`cmd.exe`):

```
echo "this is a test for windows symlink" > test_win_symlink.txt
mklink win_targetfile.txt test_win_symlink.txt
```

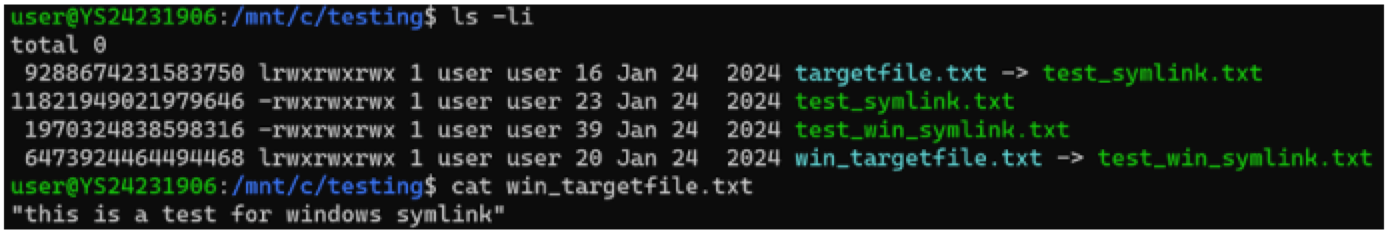
Now Windows treats this as a regular symlink that it can identify and follow:



```
C:\testing>echo "this is a test for windows symlink" > test_win_symlink.txt

C:\testing>mklink win_targetfile.txt test_win_symlink.txt
symbolic link created for win_targetfile.txt <<===>> test_win_symlink.txt

C:\testing>dir
 Volume in drive C is OS
 Volume Serial Number is E033-CB3E

 Directory of C:\testing

01/24/2024  07:07 PM    <DIR>          .
01/24/2024  02:38 PM    <JUNCTION>     targetfile.txt [...]
01/24/2024  02:38 PM                23 test_symlink.txt
01/24/2024  07:06 PM                39 test_win_symlink.txt
01/24/2024  07:07 PM    <SYMLINK>      win_targetfile.txt [test_win_symlink.txt]
               4 File(s)             62 bytes
               1 Dir(s)  642,001,002,496 bytes free

C:\testing>type win_targetfile.txt
"this is a test for windows symlink"
```

Windows can follow symlinks created on Windows.

But the Windows symlink works just as well if we access it from within WSL:



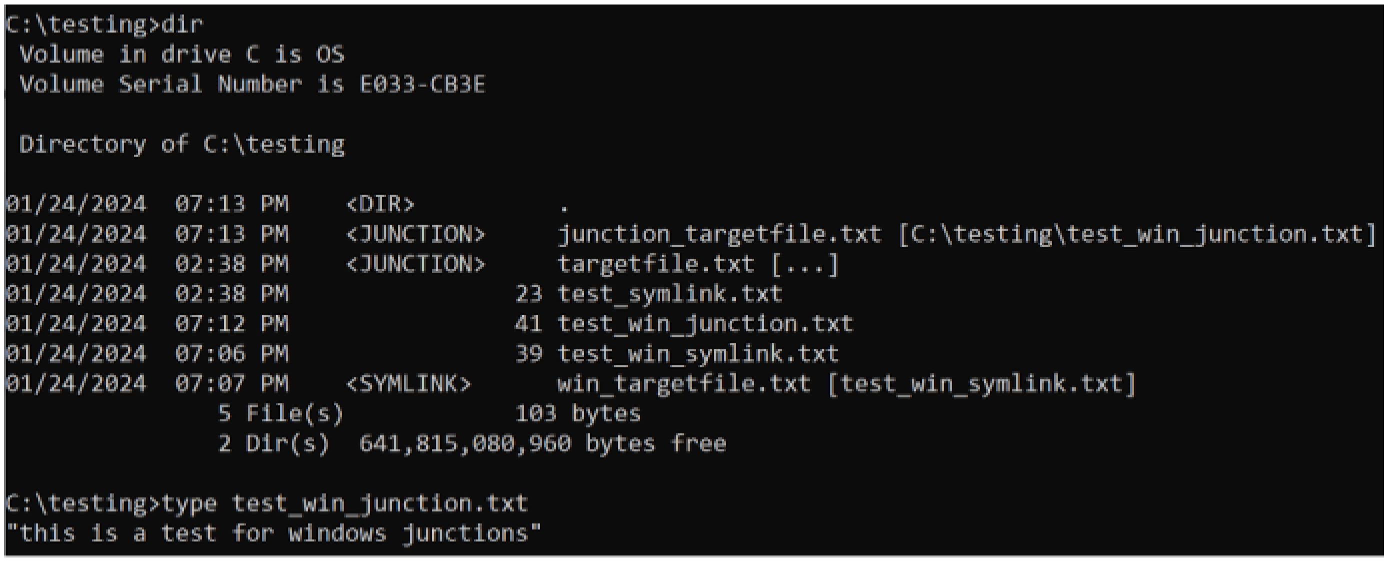
```
user@YS24231906:/mnt/c/testing$ ls -li
total 0
 9288674231583750 lrwxrwxrwx 1 user user 16 Jan 24  2024 targetfile.txt -> test_symlink.txt
11821949021979646 -rwxrwxrwx 1 user user 23 Jan 24  2024 test_symlink.txt
 1970324838598316 -rwxrwxrwx 1 user user 39 Jan 24  2024 test_win_symlink.txt
 6473924464494468 lrwxrwxrwx 1 user user 20 Jan 24  2024 win_targetfile.txt -> test_win_symlink.txt
user@YS24231906:/mnt/c/testing$ cat win_targetfile.txt
"this is a test for windows symlink"
```

The Windows symlink can also be accessed from WSL.

We get the same result if we create a file junction using the Windows command line and try to open it with WSL:

```
echo "this is a test for windows junctions" > test_win_junction.txt
mklink /J junction_targetfile.txt test_win_junction.txt
```

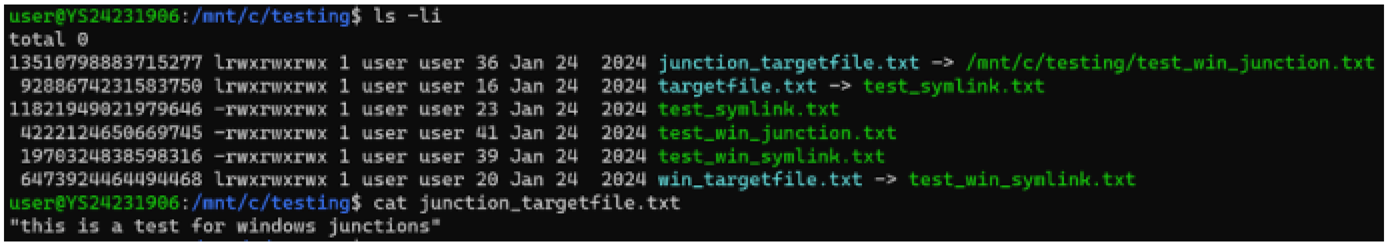
This is how the directory now looks from Windows's point of view:



```
C:\testing>dir
 Volume in drive C is OS
 Volume Serial Number is E033-CB3E

 Directory of C:\testing

01/24/2024  07:13 PM    <DIR>          .
01/24/2024  07:13 PM    <JUNCTION>     junction_targetfile.txt [C:\testing\test_win_junction.txt]
01/24/2024  02:38 PM    <JUNCTION>     targetfile.txt [...]
01/24/2024  02:38 PM                23 test_symlink.txt
01/24/2024  07:12 PM                41 test_win_junction.txt
01/24/2024  07:06 PM                39 test_win_symlink.txt
01/24/2024  07:07 PM    <SYMLINK>      win_targetfile.txt [test_win_symlink.txt]
               5 File(s)            103 bytes
               2 Dir(s)  641,815,080,960 bytes free

C:\testing>type test_win_junction.txt
"this is a test for windows junctions"
```

The directory from Windows's point of view

And this is how it looks from WSL's point of view:



```
user@YS24231906:/mnt/c/testing$ ls -li
total 0
13510798883715277 lrwxrwxrwx 1 user user 36 Jan 24  2024 junction_targetfile.txt -> /mnt/c/testing/test_win_junction.txt
 9288674231583750 lrwxrwxrwx 1 user user 16 Jan 24  2024 targetfile.txt -> test_symlink.txt
11821949021979646 -rwxrwxrwx 1 user user 23 Jan 24  2024 test_symlink.txt
 4222124650669745 -rwxrwxrwx 1 user user 41 Jan 24  2024 test_win_junction.txt
 1970324838598316 -rwxrwxrwx 1 user user 39 Jan 24  2024 test_win_symlink.txt
 6473924464494468 lrwxrwxrwx 1 user user 20 Jan 24  2024 win_targetfile.txt -> test_win_symlink.txt
user@YS24231906:/mnt/c/testing$ cat junction_targetfile.txt
"this is a test for windows junctions"
```

The directory from WSL's point of view

Hard links created by WSL do work normally on Windows, so this issue applies only to symlinks.

To summarize, Windows handles only symlinks that were created by Windows, using its standard tags, and fails to process WSL symlinks of the “LX symlink” type. However, WSL handles both types of symlinks with no issues. If you use Windows and WSL to access the same files, it’s worth paying attention to your symlinks and how they are created to avoid the same issues I ran into.

One last thing to point out is that when Bill Demirkapi tested this behavior, he noticed that Windows could follow WSL’s symlinks when they were created with a relative path but not with an absolute path. On all systems I tested, Windows couldn’t follow any symlinks created by WSL. So there is still some mystery left here to investigate.