Commons IO / IO-556

Unexpected behavior of FileNameUtils.normalize may lead to limited path traversal vulnerabilies

Details

Type:	Bug	Status:	RESOLVED
Priority:	Major	Resolution:	Duplicate
Affects Version/s:	1.1, 1.2, 1.3, 1.3.1, 1.3.2, 1.4, 2.0.1, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6	Fix Version/s:	2.7
Component/s:	Utilities		
Labels:	security security-issue		
Environment:	all		

Description

I sent this report in an Email to security@apache.org on 2017-10-16. I did not receive any kind of response yet (2017-11-18 as of writing). I am now posting it publicly, to open the issue up for discussion, and hopefully initiate a fix.

This report is not about a vulnerability in `commons-io` per se, but an unexpected behavior that has a high chance of introducing a path traversal vulnerability when using `FilenameUtils.normalize` to sanitize user input. The traversal is limited to a single out-of-bounds-stepping "/../" segment.

Reproduction

```
FilenameUtils.normalize("//../foo");        // returns "//../foo" or "\\\\..\\foo", based on java.io.File.separatorChar
FilenameUtils.normalize("\\\\..\\foo");        // returns "//../foo" or "\\\\..\\foo", based on java.io.File.separatorChar
```

Possible impact (example)

Consider a web-application that uses `FilenameUtils.normalize` to sanitize a user-supplied file name string, and then appends the sanitized value to a configured upload directory to store the uploaded content in:

```
String fileName = "//../foo";            // actually user-supplied (e.g. from multipart-POST request)
fileName = FilenameUtils.normalize(fileName);    // still holds the same value ("//../foo")

if (fileName != null) {
    File newFile = new File("/base/uploads", fileName);    // java.io.File treats double forward slashes as singles
                            // newFile now points to "/base/uploads//../foo"
    newFile = newFile.getCanonicalFile();            // newFile now points to "/base/foo", which should be inaccessible

    // Write contents to newFile...
} else {
    // Assume malicious activity, handle error
}
```

Relevant code locations

- `org.apache.commons.io.FilenameUtils#getPrefixLength` : everything between a leading "//" and the next "/" is treated as a UNC server name, and ignored in all further validation logic of `org.apache.commons.io.FilenameUtils#doNormalize` .

Discussion

One might argue that the given example is a misuse of the `FilenameUtils.normalize` method, and that everyone using it should expect absolute paths, full UNC paths, etc. to be returned by the method. Furthermore, the vulnerability can only occur due to the lax behavior of `java.io.File` .

On the other hand, I believe that the JavaDoc of `FilenameUtils.normalize` suggests to most readers, that this method is exactly what is needed to sanitize file names:

```
//-----------------------------------------------------------------------
    /**
     * Normalizes a path, removing double and single dot path steps,
     * and removing any final directory separator.
     * <p>
     * This method normalizes a path to a standard format.
     * The input may contain separators in either Unix or Windows format.
     * The output will contain separators in the format of the system.
     * <p>
     * A trailing slash will be removed.
     * A double slash will be merged to a single slash (but UNC names are handled).
     * A single dot path segment will be removed.
     * A double dot will cause that path segment and the one before to be removed.
     * If the double dot has no parent path segment to work with, {@code null}
     * is returned.
     * <p>
     * The output will be the same on both Unix and Windows except
     * for the separator character.
     * <pre>
     * /foo//               --&gt;   /foo
     * /foo/./              --&gt;   /foo
     * /foo/../bar          --&gt;   /bar
     * /foo/../bar/         --&gt;   /bar
```

I have done a quick survey of the usages of the method in public GitHub repositories. I have found numerous projects that suffer from the limited path traversal vulnerability that is described here because of this very issue. This includes Webservers, Web-Frameworks, Archive-Extraction-Software, and others.

Preventing path traversal attacks is not trivial, and many people turn to libraries like `commons-io` to avoid making mistakes when implementing parsing logic on their own. They trust that `FilenameUtils` will provide them with the most complete, and suitable sanitation logic for file names.
In addition, ".." is not a valid UNC host name according to https://msdn.microsoft.com/de-de/library/gg465305.aspx , so prohibiting it shouldn't result in any major problems.

Issue Links

duplicates

IO-559 FilenameUtils.normalize should verify hostname syntax in UNC path RESOLVED

Activity

Sravan Putluru added a comment - 11/Mar/20 17:22

Project uses **normalize()** to generated file path based on windows\linux but in VeraCode security can report method used line detected as Directory Traversal T issue as medium flaws.
Common.io 2.6 API Unexpected behavior with normalize(String s) method is not performing validations on path input. "../ " is allowing but return as Null if the input type is some thing like "**../../**". with the below lines of code checks can be remove path DT vulnerabilities issue. Could somebody please give solution.
Veracode report result Directiry Travesal medium flaws detected need to fix.

fileName = "../../etc/passwd";

fileName = FilenameUtils.normalize(fileName); // still holds the same value ("//../foo")

if (fileName != null)
{ // file creation path eg: drivec\root\06-03-2020\folder\test }

else
{ throw new CustomerException("Invalid path creation found"); }

James Howe added a comment - 21/Apr/21 13:19 - edited

Was 2.2 when this was introduced, or does it go back to 1.4 etc. as CVE-2021-29425 says?

Julius Davies added a comment - 09/Sep/21 18:46 - edited

jameshowe - goes back to 1.1.   FilenameUtils not present in 1.0.

```
java -cp commons-io-1.1.jar:. T '//../foo'
//../foo
```

```
java -cp commons-io-2.6.jar:. T '//../foo'
//../foo
```

```
java -cp commons-io-2.7.jar:. T '//../foo'
null
```

(Based on a T.java like this):

```
public class T {
  public class T { public static void main(String[] args) throws Exception {
    System.out.println( org.apache.commons.io.FilenameUtils.normalize(args[0]));
  }
}
```

Julius Davies added a comment - 09/Sep/21 19:01

I adjusted the description just now: s/FileNameUtils/FilenameUtils/g

I'm an apache committer from 2008 in "commons" but haven't been active in at least 8 years. Until today. 🙂

People

Assignee:

Unassigned

Reporter:

Lukas Euler

Votes:

0 Vote for this issue

Watchers:

5 Start watching this issue

Dates

Created:
18/Nov/17 18:27

Updated:
09/Sep/21 19:01

Resolved:
30/Nov/17 20:34