Data7 implementation

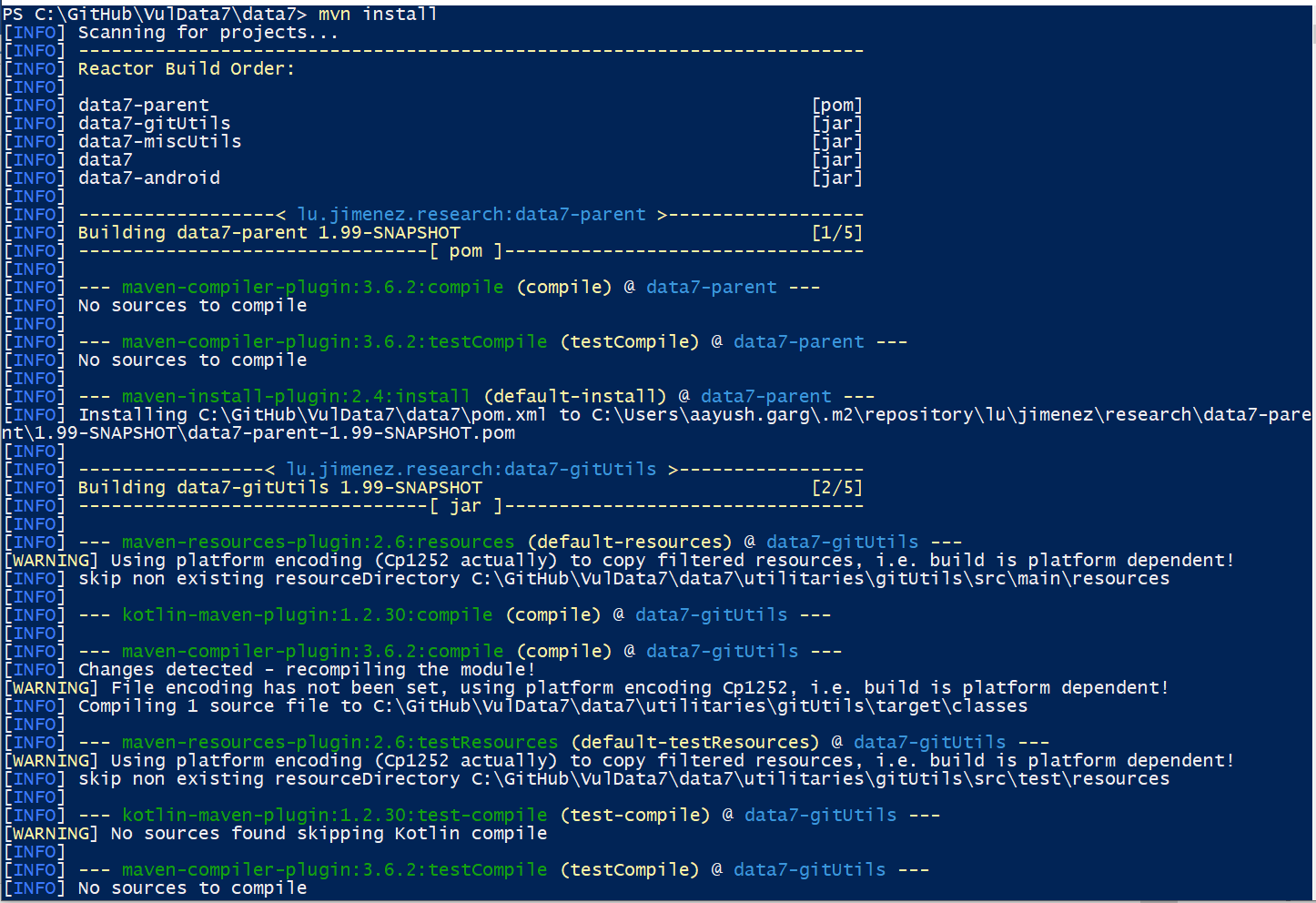
Data7 repository: <https://github.com/electricalwind/data7>

Data7 implementation (my code) repository: <https://github.com/aayushgargbu/Implementation_Data7>

Implementation Steps

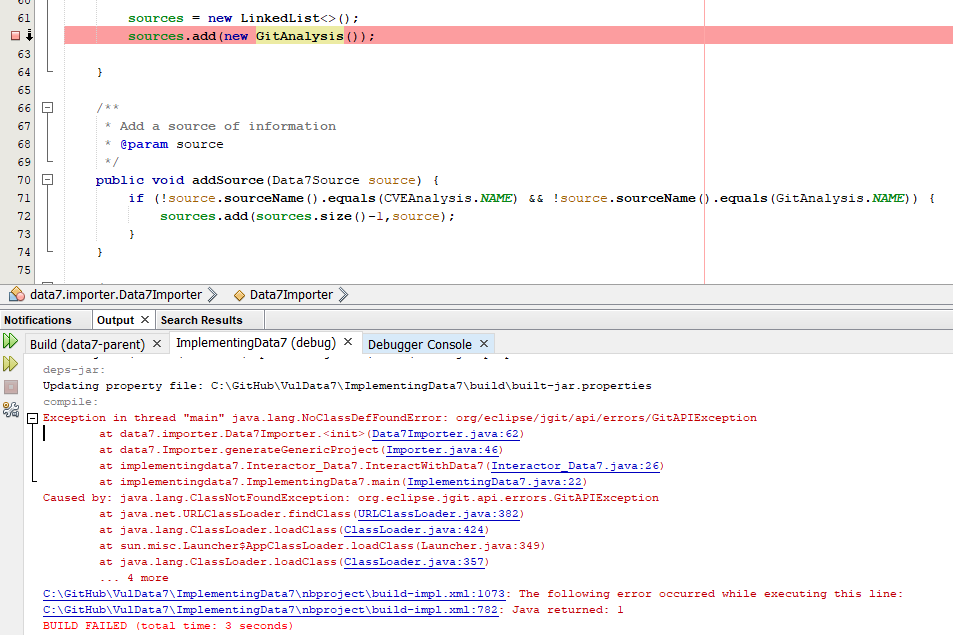
Downloaded project from github repository : <https://github.com/electricalwind/data7>

Did an mvn install as below:



Created another project (Java) to utilize the classes from above.

Getting error as below:



Seems a dependency issue. Added below in POM file and mvn install again:

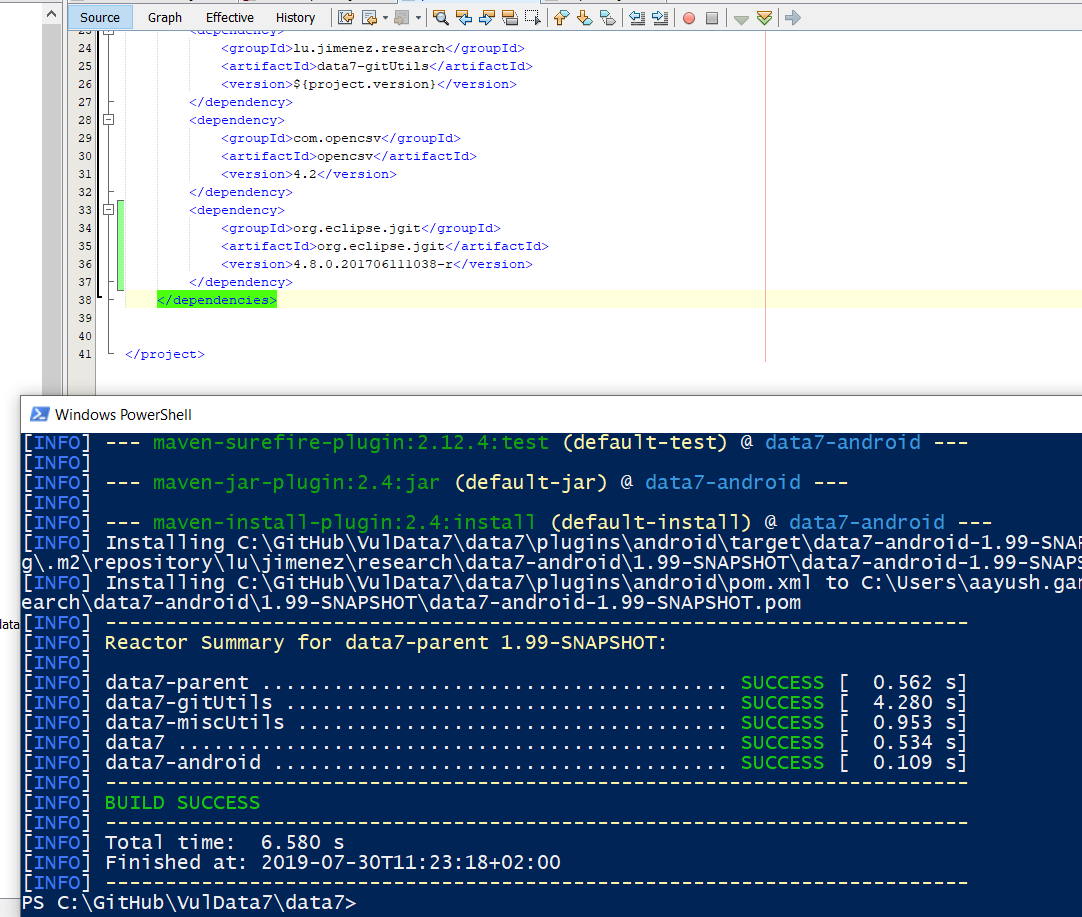
<dependency>

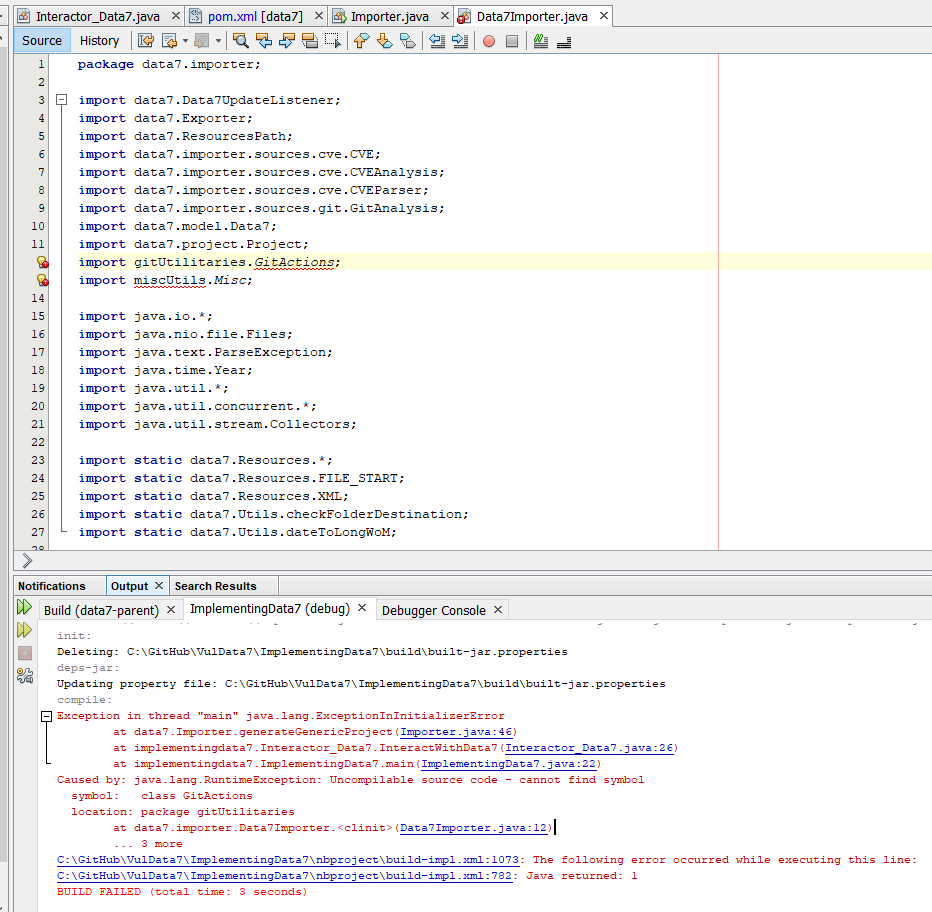
<groupId>org.eclipse.jgit</groupId>

<artifactId>org.eclipse.jgit</artifactId>

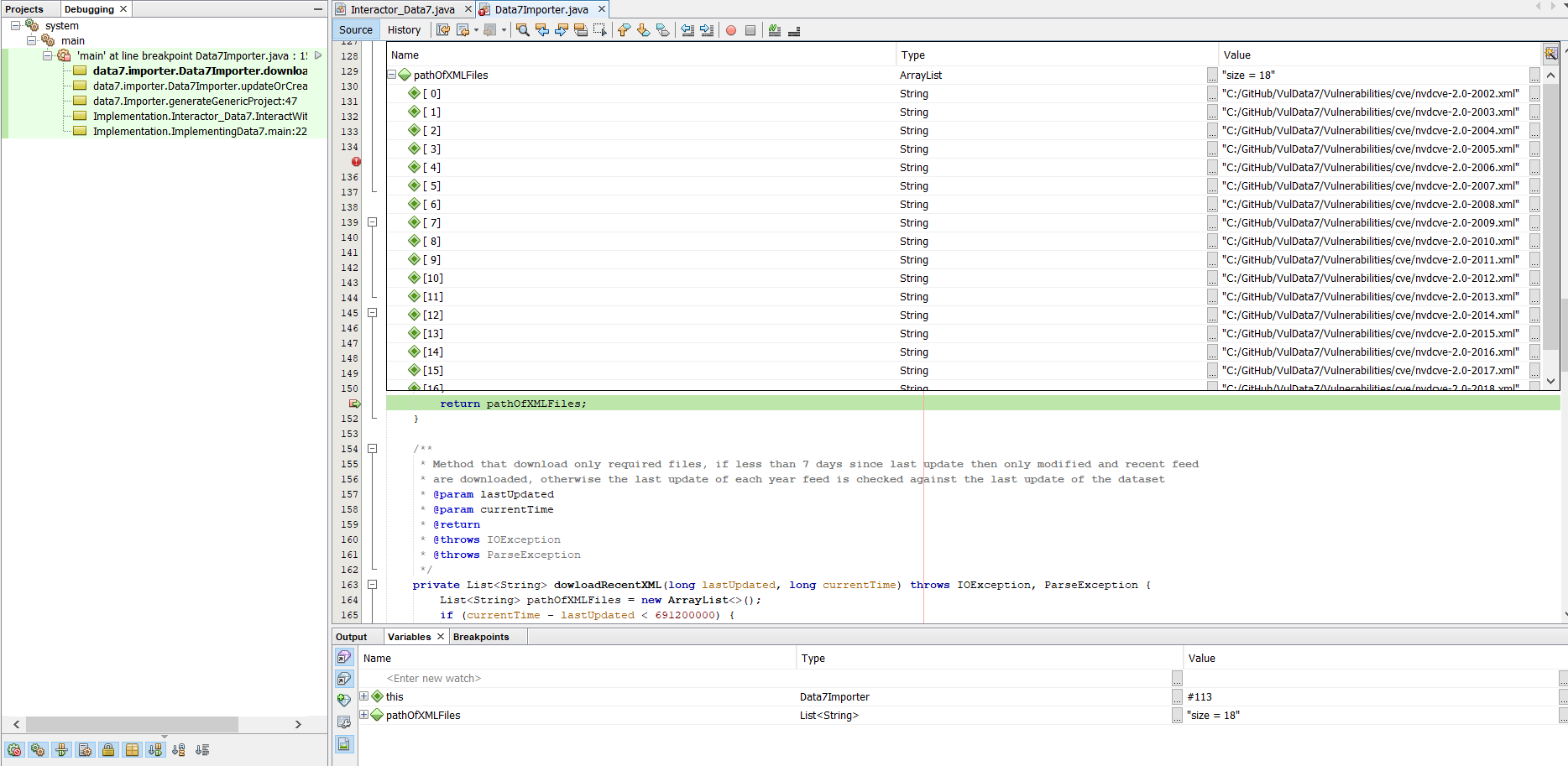
<version>4.8.0.201706111038-r</version>

</dependency>

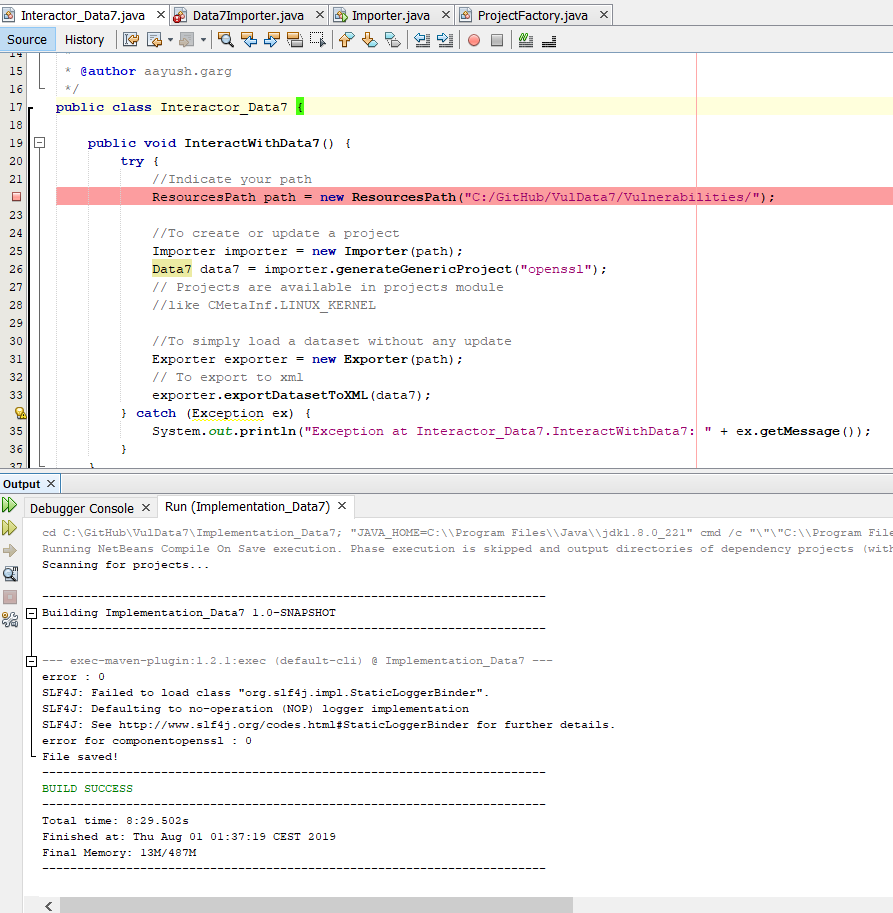


Now new error: 

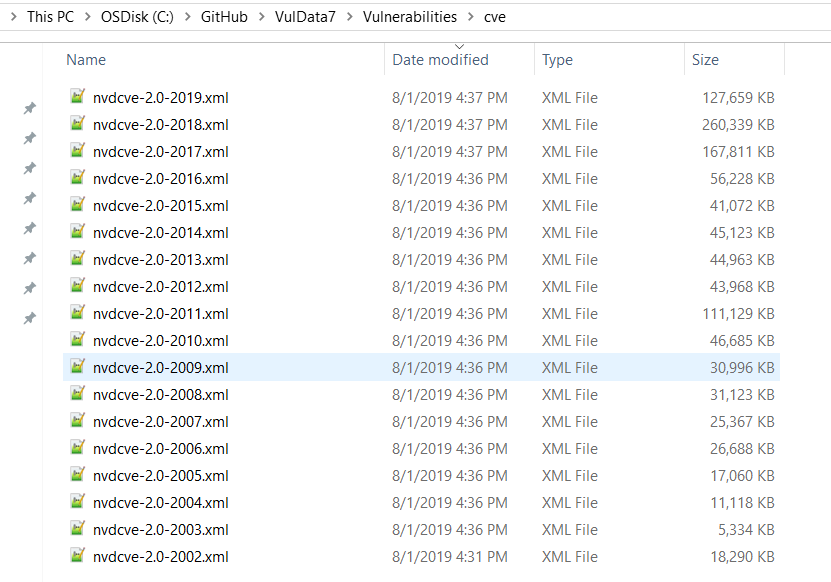
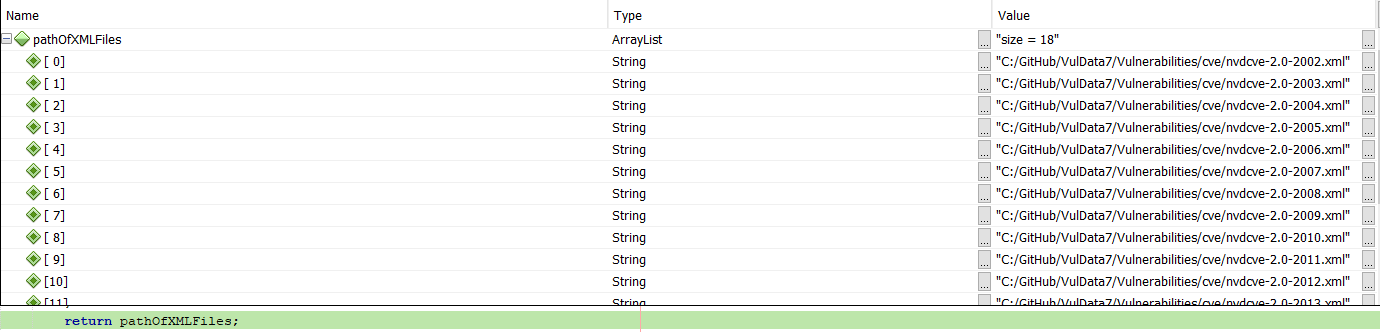
After spending two days trying to solve the issue, it was realized that the change is required in the project implementing the Data7 rather than in the Data7 POM files instead. Due to unavailability of POM file. A new MAVEN Java project was created and entire process was done again. But this time it just worked.



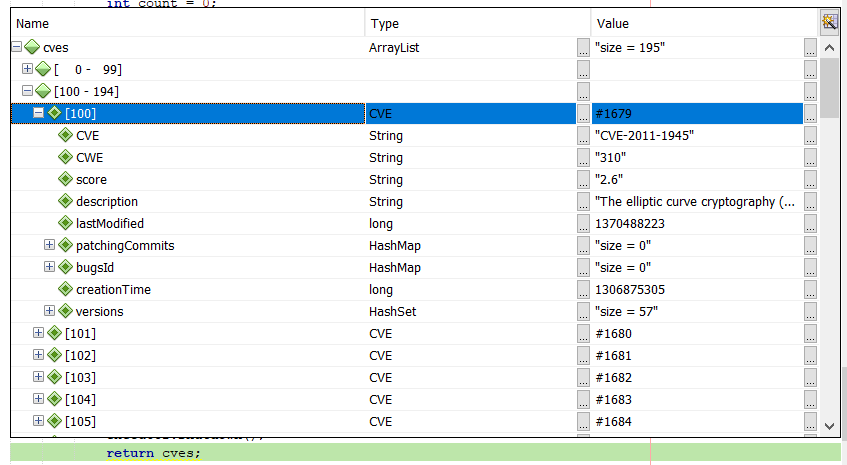
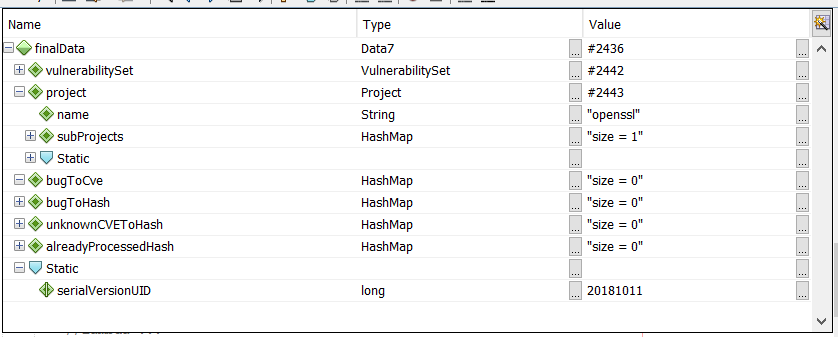
Success

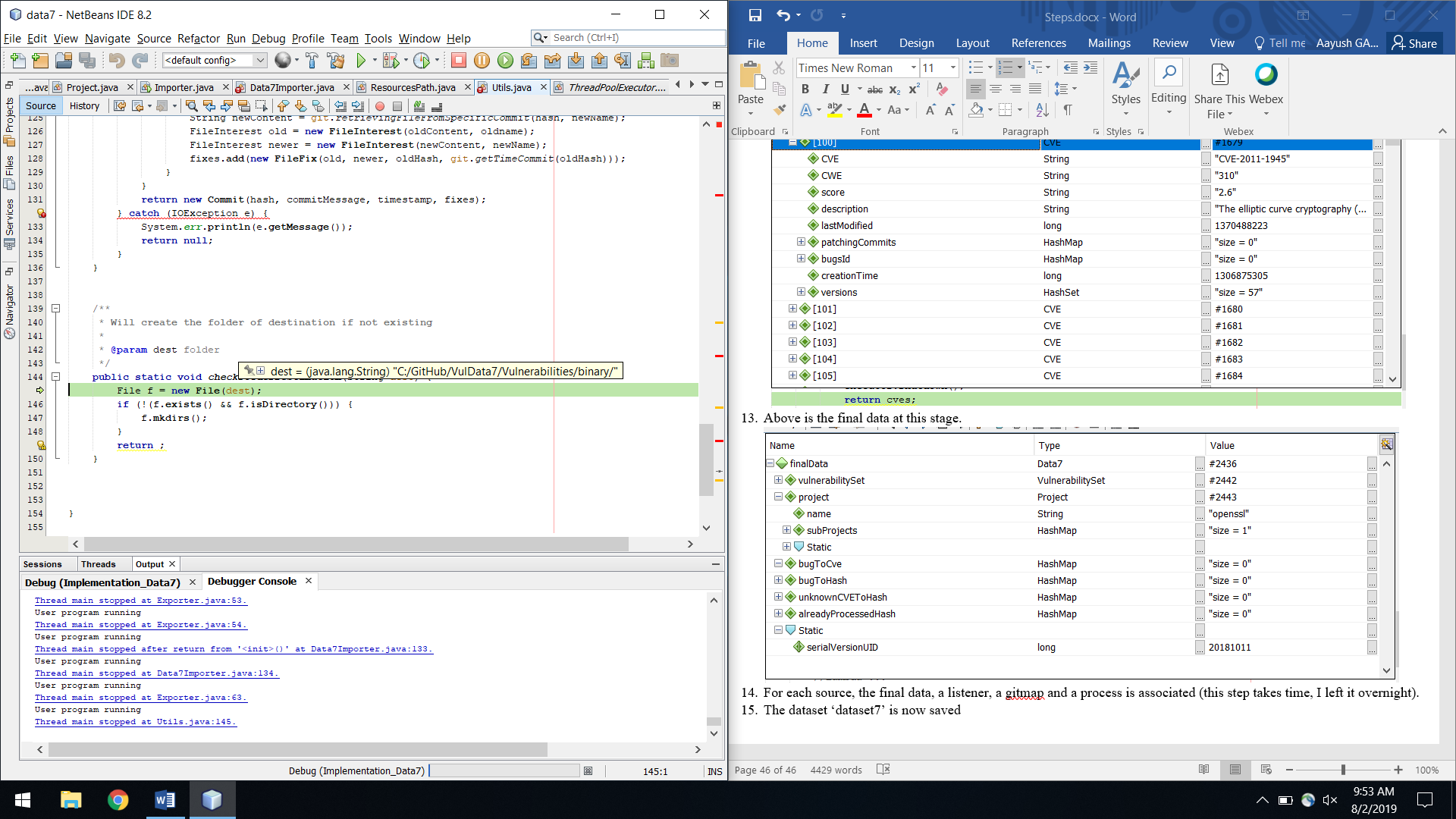


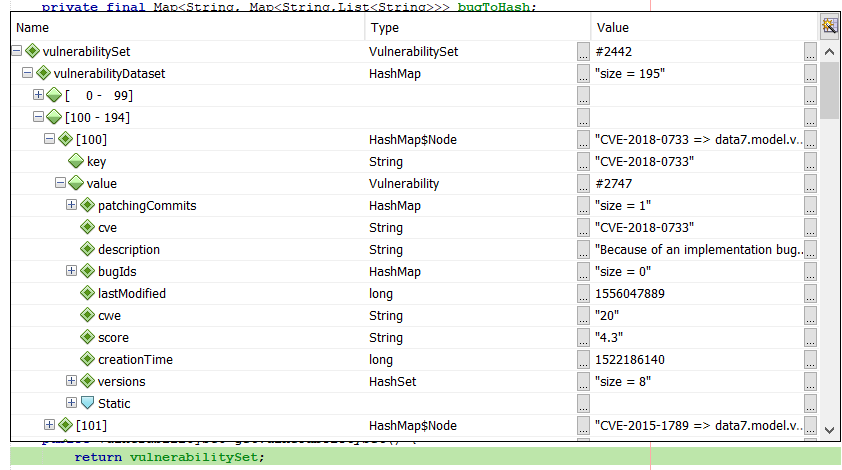
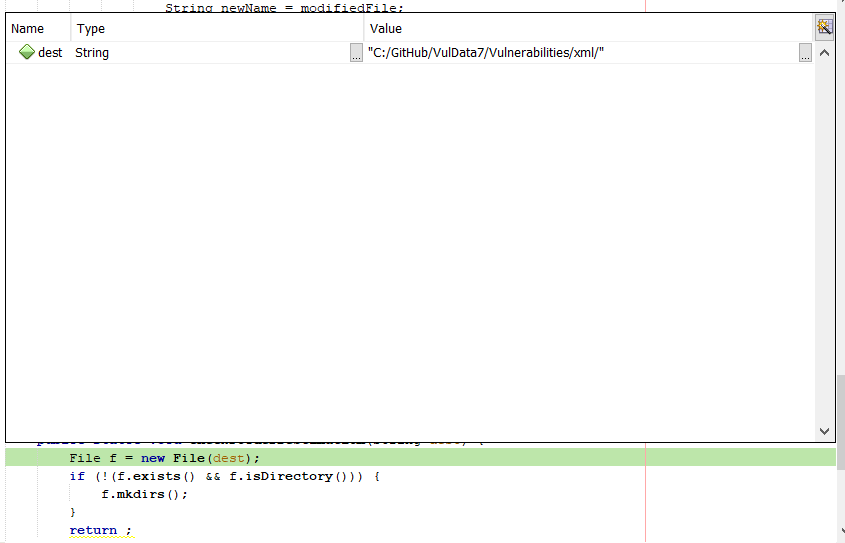
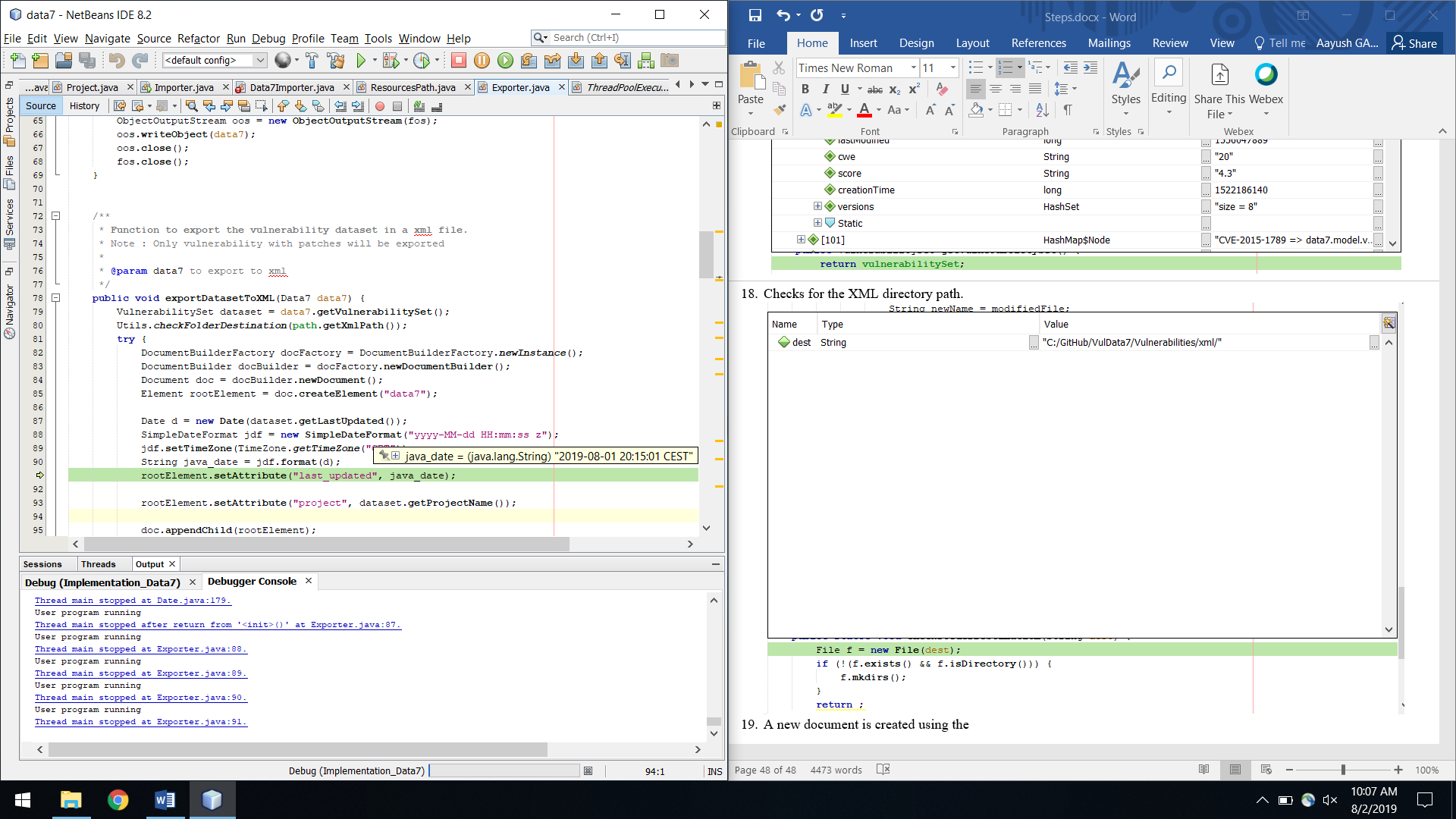
Data7 Code/Working overview

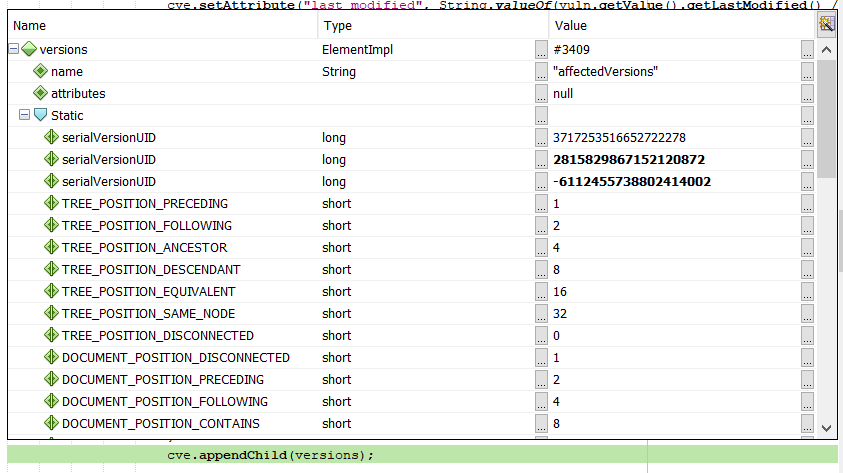
1. Checks for the path in format: C:/GitHub/VulData7/Vulnerabilities/ and sets the internal directory structure for binary, git, xml and cve. If no directory found gives error “Path is incorrect or inexisting”
2. Checks for the user input project name and retrieves project info (linux, openssl, wireshark, systemd, android). Returns Project object with name and subprojects name. e.g. OPENSSL\_NVD,OPEN\_SSL. If no match is found exception will be Project is null (next step)
3. Now we have the path, project and listeners, new GitAnalysis instance is added to sources.
4. It now checks for -data7.obj file. If not, Step 5
5. If does not exist returns null and creates addresses of files (xml) to download from 2002 to this year.
6. Downloads the files with path format: CVE\_URL + year + XML, path.getCvePath()
7. .zip is replaced with ‘’
8. Above step downloads files in cve folder 
9. Returns path of files as 

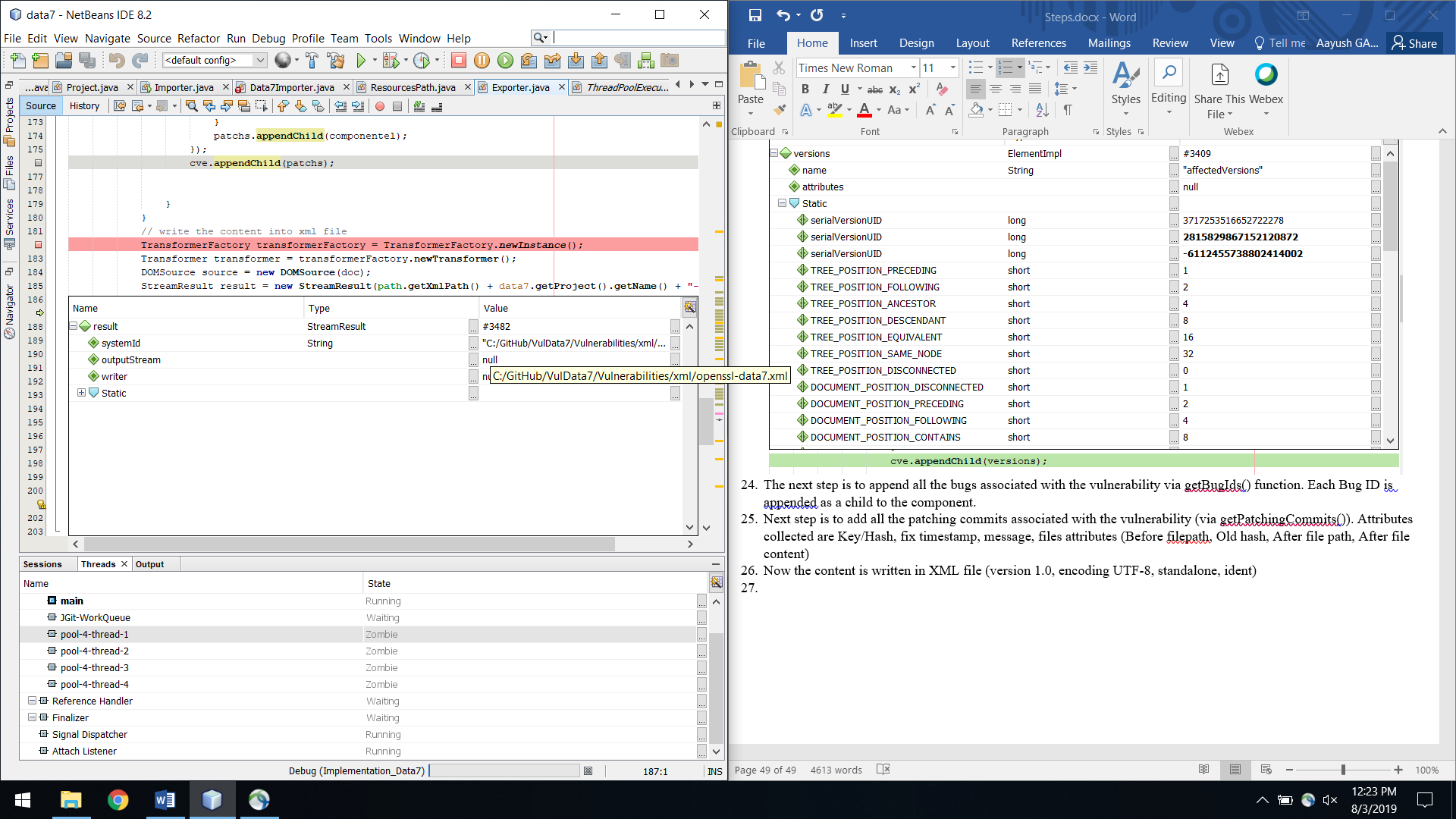
Note: Sometimes static.nvd.nist.gov is down which makes the project unable to perform, as the connection to the website is the starting point.

1. New Project type object is created with few keyValue pairs.
2. A thread is created NB\_THREADS to queue all the XMLs (2002 – 2019, in our case) for parsing.
3. The parsed results are added to list of CVEs, and the executor shuts down after that. 
4. Above is the final data at this stage. 
5. For each source, the final data, a listener, a gitmap and a process is associated (this step takes time, I left it overnight).
6. The dataset ‘dataset7’ is now saved as below



1. Folders are checked for existence, otherwise created and data written via ObjectOutputStream
2. Exporter object in main class – Starts with getting vulnerabilitySet (the one collected earlier) 
3. Checks for the XML directory path. 
4. A new document is created, meanwhile date last updated is checked for. 
5. For each vulnerability, if there are patching commits (the size of patching commits>0), cwe, if available, is appended to the cve.
6. The score of the vulnerability is appended to CVE.
7. Next step is to append the description of vulnerability
8. Next is to add all the components of version to the CVE as below:



1. The next step is to append all the bugs associated with the vulnerability via getBugIds() function. Each Bug ID is appended as a child to the component.
2. Next step is to add all the patching commits associated with the vulnerability (via getPatchingCommits()). Attributes collected are Key/Hash, fix timestamp, message, files attributes (Before filepath, Old hash, After file path, After file content)
3. Now the content is written in XML file (version 1.0, encoding UTF-8, standalone, ident) as below:
4. Output is openssl-data7.xml. User program finished.