**CSCI 5742-Cybersecurity Programming-Lab 05-XML and Docx in Python**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Goals:

* Understand how to parse XML in Python
* Understand how to create a MS Word, docx format in Python

Deliverables

Fully commented parsexml.py, screenshot of parsexml, fully commented docx.py, docxoutput.docx. Note there are no “thought questions” so this lab does not need to be turned in.

Cybersecurity consulting is all about information. Much of the information that is output from tools, has an option to be output in XML format. Then once XML format, cybersecurity analysts still have a need to compile this information along with information gained from other sources. Then the cybersecurity analyst is often asked to compile this information into a useable format (usually Microsoft docx format)

**Part I - XML Parsing in python**

* This task in Kali is just to show you that many of the tools have an option for XML output. Unfortunately each tool has its own format, so consolidating tools output requires some amount of coding.
* Open the Kali 2018.2. in Host Only network with student, PW: Student123
* Open a terminal
* **su**
* **service apache2 start**
* **nmap -oX nmaptest.xml 127.0.0.1** (that's a little oh)
* **more nmaptest.xml** (space to go a page at a time)
* Notice the tag <scaninfo type="syn" protocol="tcp" numservices = "1000" services="… "/> This says that we did a synchronize scan…sending syn packets, and awaiting the ack packets. Our protocol was tcp, and we looked at up to 1000 ports, as listed in services
* Now that you understand how tools can output their information in XML format, we will now have you program in python 3.6, using any IDE that you choose. The results do not require a connection. You can shut down your Kali 2.2 (gracefully) if you are done with it.
* **While the following tasks should logically go before Part II - If you do not have Microsoft Word on your mobile system, you should do Part II first, while you have access to Microsoft Word on the lab machines**
* We will be using Python 3.6+. You have the choice of OS and/or IDE. So, you can use the command line or PyCharm built into the images, on your classroom Mac desktop or on your own laptop
* Create a testxml.xml file with the following

*<?***xml version="1.0"***?>*<**doc**>  
 <**branch name="First Name Option" hash="First Hash Option"**>  
 First Text  
 </**branch**>  
 <**branch name="Second Name Option" hash="Second Name Hash Option"**>  
 Second Text  
 <**sub-branch name="Second Name Sub-branch"**>  
 Second sub-Text  
 </**sub-branch**>  
 </**branch**>  
 <**branch name="Third Name Option"**>  
 Third Text  
 </**branch**>  
 <**branch name="Fourth Name Option"**>  
 Fourth Text  
 <**sub-branch name="Fourth Name Sub-branch"**>  
 Fourth sub-text  
 </**sub-branch**>  
 </**branch**>  
</**doc**>

* There are many libraries that you could use to parse XML. Once parsed, you can add to databases, add to JSON type objects, add to reports etc. We will start with xml.etree as it is standard with python3
* Create a file called **parsexml.py**
* Note if you are using pycharm, you will want to Run->Edit Configurations and change the script parameters. or just run python3 from the terminal, which would be more efficient
* from the command prompt you will have to **pip3 install xmltodict**
* Set up a tree object and print out the root of the tree

**import sys  
import xml.etree.ElementTree as ET  
from xml.dom.minidom import parseString  
import json  
import xmltodict  
  
  
if (sys.argv[1]=="-h" or sys.argv[1]=="--help"):  
 print("USAGE: python3 {} inputfile.xml ".format(sys.argv[0]))**

**sys.exit()**

**fileToParse = sys.argv[1]  
print('File to Parse:{}'.format(fileToParse))  
tree = ET.ElementTree(file=fileToParse)  
print("\*\*\*\*\*")  
print("Tree Root:")  
root = tree.getroot()  
print("root tag:{}, root attribute:{}".format(root.tag, root.attrib))**

* Run this. Make sure you understand what it is doing. Translate the output into appropriate comments in the code. You MUST fully document your code for full credit.
* Now we want to print out all of the children of the root. Note that we could print out children of any tag

**print("\*\*\*\*\*")  
print("Children of root")  
for element in root:  
 print (element.tag, element.attrib, element.keys(), element.text)**

* Run this. Translate the output into appropriate comments in the code
* Ensure you know the difference between a tag, and attrib and a key
* And let's print out a few specific elements

**print("\*\*\*\*\*")  
itemsInTree = len(root)  
print("Print Indexed Items")  
print("First Item tag: {}, First Item attribute: {}".format(root[0].tag, root[0].attrib))  
print("Second Item tag: {}, Second Item attribute: {}".format(root[1].tag, root[1].attrib))  
print("Last Item tag: {}, Last Item attribute: {}".format(root[itemsInTree-1].tag, root[itemsInTree-1].attrib) )**

* Run this. Translate the output into appropriate comments
* you can also use some built in functions to find all of a tag or sub-tag

**print("\*\*\*\*\*")  
print("Iterate through entire tree, providing a list of branch/sub-branch tags")  
for element in tree.findall('branch/sub-branch'):  
 print (element.tag, element.attrib)**

* Let's delete an entire item from the tree

**print("\*\*\*\*\*")  
print("Deleting second item")  
root.remove(root[1])**

* Now add an item to the bottom of the tree

**print("\*\*\*\*\*")  
print("Insert an item at the end")  
print ("Last Item:")  
  
newElement = ET.Element('branch')  
newElement.set('name', "Newly Added Name")  
newElement.text="This is a test"  
root.append(newElement)**

* Now you should print out the tree and validate the delete and the add
* All of this is good information, but what if you want to print out a nicely formatted XML tree to a file? For that you will need to convert the Etree to a string, then use the minidom to parse the string. At the top:

**from xml.dom.minidom import parseString**

* then at the bottom of the file

**outfile = open('testxmlout.xml', 'w')  
print("\*\*\*\*\*")  
print("Print a formatted tree (reflecting changes)")  
xmlstrPretty = parseString(ET.tostring(root)).toprettyxml(indent=" ", newl=' ')  
print (xmlstrPretty)  
outfile.write(xmlstrPretty)**

**print("\*\*\*\*\*")  
print("Convert to Dict")  
xmlFile = open('doc1.xml', "rb") # notice the "rb" mode  
dictObject = xmltodict.parse(xmlFile, xml\_attribs=True)  
print(dictObject)  
print (dictObject['doc']['branch'][0]['#text'])  
  
for branch in dictObject['doc']['branch']:  
 print (branch['@name'], "\t", branch['#text'])  
  
print("\*\*\*\*\*")  
print("Convert to json and back to a dictionary")  
jsonObject = json.dumps(dictObject, indent=4)  
dictJsonObject = json.loads(jsonObject)  
print(dictJsonObject)  
print(dictJsonObject['doc']['branch'][0]['#text'])**

**Part II - Word .docx format**

You could use the above information to parse individual data components from tools or databases in XML format. Then you could add the information to an SQL database or convert to JSON objects. Let's say you did all of that and now need to make a report in Word Format. For ease of understanding, we did not incorporate the results of part one into a .docx format, but this might be a good practice exercise in the future.

It turns out that the x in docx, xlsx, or pptx format stands for OpenXML. You can actually build your own, based on an open standard, but it would probably be better to use a library.

I found a very good reference for Doc files and PDF in Automate the Boring Stuff with Python. Again, as always, please type in vs just copying and pasting so you can get a better understanding

* from the command line, find where your 3.6 pip is located (in linux or mac you can type in which pip
* run that pip with this command or pip3 depending on how it is set up
* **pip install python-docx**
* Create a **docxinput.docx** file in MS Word with a 3 lines of stuff followed by returns. Then download the banner graphic from the cse.ucdenver.edu website
* Then create a docx.xml and type the following:

**import docx  
  
#read in an existing doc**

**doc1 = docx.Document('docxinput.docx')  
  
for para in range (0, len(doc1.paragraphs)):  
 print(doc1.paragraphs[para].text)**

**#create a new doc  
doc = docx.Document()  
doc.add\_picture('ucdenvercse.png',width=docx.shared.Inches(3))  
doc.add\_paragraph('Your first and Last Name')  
paraObj1 = doc.add\_paragraph('This is a second paragraph.')  
paraObj1.add\_run(' This text is being added to the second paragraph.')  
  
doc.add\_heading('Header 1', 1)  
doc.add\_heading('Header 2', 2)  
doc.add\_heading('Header 3', 3)  
  
# populate dataset  
recordset = [  
 {  
 "id" : 1,  
 "qty": 2,  
 "desc": "Thing 1"  
 },  
 {  
 "id" : 3,  
 "qty": 4,  
 "desc": "Thing 2"  
 },  
]  
  
# add table ------------------  
table = doc.add\_table(1, 3) #1 row, 3 columns  
  
# populate header row --------  
table = doc.add\_table(rows=1, cols=3, style = 'Table Grid')  
#table.style = 'Table Grid'  
hdr\_cells = table.rows[0].cells  
hdr\_cells[0].text = 'Qty'  
hdr\_cells[1].text = 'Id'  
hdr\_cells[2].text = 'Desc'  
for item in recordset:  
 row\_cells = table.add\_row().cells  
 row\_cells[0].text = str(item["qty"])  
 row\_cells[1].text = str(item["id"])  
 row\_cells[2].text = item["desc"]  
  
para = doc.add\_paragraph()  
para.add\_run('This is on the')  
boldItalic=para.add\_run(' first')  
boldItalic.bold = True  
boldItalic.italic = True  
para.add\_run(' page!')  
  
currentPara = len(doc.paragraphs)  
doc.paragraphs[currentPara-1].runs[2].add\_break(docx.enum.text.WD\_BREAK.PAGE)  
para = doc.add\_paragraph()  
  
para.add\_run('This is on the')  
boldItalic=para.add\_run(' second')  
boldItalic.bold = True  
boldItalic.italic = True  
para.add\_run(' page!')  
  
doc.save('docxoutput.docx')**

Turn in the deliverables noted at the top of this document. If you happen to finish early, please use the time to work on your final project.