Code:

vulscan.py

#!/usr/bin/env python

"""

Vulscan

"""

from \_\_future\_\_ import with\_statement

import time, base64

import filecmp

import os,sys,re,random

from StringIO import StringIO

try:

from lxml import etree

except ImportError:

try:

import xml.etree.cElementTree as etree

except ImportError:

try:

import xml.etree.ElementTree as etree

except ImportError:

print "Cannot find the ElementTree in your python packages"

\_\_application\_\_ = "vulscan"

\_\_version\_\_ = "0.1"

\_\_release\_\_ = \_\_application\_\_ + '/' + \_\_version\_\_

\_\_author\_\_ = "Sagar Palao"

names = {

'xss' : 'Cross-Site Scripting',

'sqli' : 'SQL Injection',

'csrf' : 'Cross-Site Request Forgery',

'dos' : 'Denial Of Service',

'dt' : 'Directory Traversal',

'spam' : 'Spam',

'id' : 'Information Disclosure',

'rfe' : 'Remote File Execution',

'lfi' : 'Local File Inclusion'

}

c\_reg = re.compile(r'^(.+)-(.\*)\[(.+)[-|+](\d+)\] "([A-Z]+)?(.+) HTTP/\d.\d" (\d+)(\s[\d]+)?(\s"(.+)" )?(.\*)$')

table = {}

txt\_header = """

#

# File created by Vulscan after log analysis on http services

"""

xml\_header = """<!--

File created by Vulscan

-->

<?xml version="1.0" encoding="utf-8"?>

"""

html\_header = """

<html>

<head>

<link rel = "stylesheet" href="./style/report.css" />

<link rel="stylesheet" href="./style/jQuery-plugin-progressbar.css">

<script src="./style/jquery-3.1.1.js"></script>

<script src="./style/jQuery-plugin-progressbar.js"></script>

</head>

<body>

<div class="bg"></div>

<div class="bg2"></div>

<div class="bg3"></div>

<div class="part1">Generic Threat</div>

<div class="part2">Specific Threat</div>

"""

html\_footer = "<div id='footer'>Vulscan</div></body></html>"

class object\_dict(dict):

def \_\_init\_\_(self, initd=None):

if initd is None:

initd = {}

dict.\_\_init\_\_(self, initd)

def \_\_getattr\_\_(self, item):

d = self.\_\_getitem\_\_(item)

# if value is the only key in object, you can omit it

if isinstance(d, dict) and 'value' in d and len(d) == 1:

return d['value']

else:

return d

def \_\_setattr\_\_(self, item, value):

self.\_\_setitem\_\_(item, value)

def \_\_parse\_node(node):

tmp = object\_dict()

if node.text:

tmp['value'] = node.text

for (k,v) in node.attrib.items():

tmp[k] = v

for ch in node.getchildren():

cht = ch.tag

chp = \_\_parse\_node(ch)

if cht not in tmp:

tmp[cht] = chp

continue

old = tmp[cht]

if not isinstance(old, list):

tmp.pop(cht)

tmp[cht] = [old]

tmp[cht].append(chp)

return tmp

def parse(xml\_file):

try:

xml\_handler = open(xml\_file, 'r')

doc = etree.parse(xml\_handler).getroot()

xml\_handler.close()

return object\_dict({doc.tag: \_\_parse\_node(doc)})

except IOError:

print "error: problem with the filter's file"

return {}

def get\_value(array, default):

if 'value' in array:

return array['value']

return default

def html\_entities(str):

out = ""

for i in str:

if i == '"': out += '&quot;'

elif i == '<': out += '&lt;'

elif i == '>': out += '&gt;'

else:

out += i

return out

d\_replace = {

"\r":";",

"\n":";",

"\f":";",

"\t":";",

"\v":";",

"'":"\"",

"+ACI-":"\"",

"+ADw-":"<",

"+AD4-" : ">",

"+AFs-" : "[",

"+AF0-" : "]",

"+AHs-" : "{",

"+AH0-" : "}",

"+AFw-" : "\\",

"+ADs-" : ";",

"+ACM-" : "#",

"+ACY-" : "&",

"+ACU-" : "%",

"+ACQ-" : "$",

"+AD0-" : "=",

"+AGA-" : "'",

"+ALQ-" : "\"",

"+IBg-" : "\"",

"+IBk-" : "\"",

"+AHw-" : "|",

"+ACo-" : "\*",

"+AF4-" : "^",

"+ACIAPg-" : "\">",

"+ACIAPgA8-" : "\">",

}

re\_replace = None

def fill\_replace\_dict():

global d\_replace, re\_replace

# very first control-chars

for i in range(0,20):

d\_replace["%%%x" % i] = "%00"

d\_replace["%%%X" % i] = "%00"

# javascript charcode

for i in range(33,127):

c = "%c" % i

d\_replace["\\%o" % i] = c

d\_replace["\\%x" % i] = c

d\_replace["\\%X" % i] = c

d\_replace["0x%x" % i] = c

d\_replace["&#%d;" % i] = c

d\_replace["&#%x;" % i] = c

d\_replace["&#%X;" % i] = c

# SQL words?

d\_replace["is null"]="=0"

d\_replace["like null"]="=0"

d\_replace["utc\_time"]=""

d\_replace["null"]=""

d\_replace["true"]=""

d\_replace["false"]=""

d\_replace["localtime"]=""

d\_replace["stamp"]=""

d\_replace["binary"]=""

d\_replace["ascii"]=""

d\_replace["soundex"]=""

d\_replace["md5"]=""

d\_replace["between"]="="

d\_replace["is"]="="

d\_replace["not in"]="="

d\_replace["xor"]="="

d\_replace["rlike"]="="

d\_replace["regexp"]="="

d\_replace["sounds like"]="="

re\_replace = re.compile("(%s)" % "|".join(map(re.escape, d\_replace.keys())))

def multiple\_replace(text):

return re\_replace.sub(lambda mo: d\_replace[mo.string[mo.start():mo.end()]], text)

def decode\_attempt(string):

return multiple\_replace(string)

def analyzer(data):

exp\_line, regs, array, preferences, org\_line = data[0],data[1],data[2],data[3],data[4]

done = []

for attack\_type in preferences['attack\_type']:

if attack\_type in regs:

if attack\_type not in array:

array[attack\_type] = {}

for \_hash in regs[attack\_type]:

if \_hash not in done:

done.append(\_hash)

attack = table[\_hash]

cur\_line = exp\_line[5]

if preferences['encodings']:

cur\_line = decode\_attempt(cur\_line)

if attack[0].search(cur\_line):

if attack[1] not in array[attack\_type]:

array[attack\_type][attack[1]] = []

array[attack\_type][attack[1]].append((exp\_line, attack[3], attack[2], org\_line))

if preferences['exhaustive']:

break

else:

return

def Vulscaner(access, filters, preferences = [], output = 'text'):

print output

global table

if not os.path.isfile(access):

print "error: the log file doesn't exist"

return

if not os.path.isfile(filters):

print "error: the filters file (XML) doesn't exist"

print "please download the filter"

return

if output not in ('html', 'text', 'xml'):

print "error: the output format '%s' hasn't been recognized" % output

return

xml\_filters = parse(filters)

len\_filters = len(xml\_filters)

if len\_filters < 1:

return

regs = {}

print "Loading XML file '%s'..." % filters

for group in xml\_filters:

for f in xml\_filters[group]:

if f == 'filter':

if type(xml\_filters[group][f]) == type([]):

for elmt in xml\_filters[group][f]:

rule, impact, description, tags = "",-1,"",[]

if 'impact' in elmt:

impact = get\_value(elmt['impact'], -1)

if 'rule' in elmt:

rule = get\_value(elmt['rule'], "")

if 'description' in elmt:

description = get\_value(elmt['description'], "")

if 'tags' in elmt and 'tag' in elmt['tags']:

if type(elmt['tags']['tag']) == type([]):

for tag in elmt['tags']['tag']:

tags.append(get\_value(tag, ""))

else:

tags.append(get\_value(elmt['tags']['tag'], ""))

for t in tags:

compiled = None

if t not in regs:

regs[t] = []

try:

compiled = re.compile(rule)

except Exception:

print "The rule '%s' cannot be compiled properly" % rule

return

\_hash = hash(rule)

if impact > -1:

table[\_hash] = (compiled, impact, description, rule, \_hash)

regs[t].append(\_hash)

if len(preferences['attack\_type']) < 1:

preferences['attack\_type'] = regs.keys()

flag = {}

print "Processing the file '%s'..." % access

sample, sampled\_lines = False, []

if preferences['sample'] != float(100):

sample = True

total\_nb\_lines = sum(1 for line in open(access))

random.seed(time.clock())

sampled\_lines = random.sample(range(total\_nb\_lines), int(float(total\_nb\_lines) \* preferences['sample'] / float(100)))

sampled\_lines.sort()

loc, lines, nb\_lines = 0, 0, 0

old\_diff = 0

start = time.time()

diff = []

with open(access) as log\_file:

for line in log\_file:

lines += 1

if sample and lines not in sampled\_lines:

continue

if c\_reg.match(line):

out = c\_reg.search(line)

ip = out.group(1)

name = out.group(2)

date = out.group(3)

ext = out.group(4)

method = out.group(5)

url = out.group(6)

response = out.group(7)

byte = out.group(8)

referrer = out.group(9)

agent = out.group(10)

if not correct\_period(date, preferences['period']):

continue

loc += 1

if len(url) > 1 and method in ('GET','POST','HEAD','PUT','PUSH','OPTIONS'):

analyzer([(ip,name,date,ext,method,url,response,byte,referrer,agent),regs,flag, preferences, line])

elif preferences['except']:

diff.append(line)

if nb\_lines > 0 and lines > nb\_lines:

break

tt = time.time() - start

n = 0

for t in flag:

for i in flag[t]:

n += len(flag[t][i])

print "Vulscan results:"

print "\tProcessed %d lines over %d" % (loc,lines)

print "\tFound %d attack patterns in %f s" % (n,tt)

short\_name = access[access.rfind(os.sep)+1:]

hcmp = filecmp.cmp('/etc/hosts', './template\_files/hosts\_temp')

nscmp = filecmp.cmp('/etc/nsswitch.conf', './template\_files/nsswitch\_temp')

rcomp = filecmp.cmp('/etc/resolv.conf', './template\_files/resolv\_temp')

vr = 0

if hcmp == True:

vr = vr + 0.34

if nscmp == True:

vr = vr + 0.33

if rcomp == True:

vr = vr + 0.33

#if n > 0:

print "Generating output in %s%s%s\_Vulscan\_\*" % (preferences['odir'],os.sep,short\_name)

if 'html' in preferences['output']:

generate\_html\_file(flag, short\_name, filters, preferences['odir'], vr)

elif 'text' in preferences['output']:

generate\_text\_file(flag, short\_name, filters, preferences['odir'])

elif 'xml' in preferences['output']:

generate\_xml\_file(flag, short\_name, filters, preferences['odir'])

if len(diff) > 0:

o\_except = open(os.path.abspath(preferences['odir'] + os.sep + "Vulscan\_except.txt"), "w")

for l in diff:

o\_except.write(l + '\n')

o\_except.close()

def generate\_text\_file(flag, access, filters, odir):

curtime = time.strftime("%a-%d-%b-%Y", time.localtime())

fname = '%s\_Vulscan\_%s.txt' % (access, curtime)

fname = os.path.abspath(odir + os.sep + fname)

try:

out = open(fname, 'w')

out.write(txt\_header)

out.write("Vulscaned file: %s\n" % access)

out.write("Creation date: %s\n\n" % curtime)

for attack\_type in flag:

if attack\_type in names:

out.write("Attack %s (%s)\n" % (names[attack\_type], attack\_type))

else:

out.write("Attack type: %s\n" % attack\_type)

impacts = flag[attack\_type].keys()

impacts.sort(reverse=True)

for i in impacts:

out.write("\n\t### Impact %d\n" % int(i))

for e in flag[attack\_type][i]:

out.write("\t%s" % e[3])

out.write("\tReason: \"%s\"\n\n" % e[2])

out.close()

except IOError:

print "Cannot open the file:", fname

return

def generate\_xml\_file(flag, access, filters, odir):

curtime = time.strftime("%a-%d-%b-%Y", time.localtime())

fname = '%s\_Vulscan\_%s.xml' % (access, curtime)

fname = os.path.abspath(odir + os.sep + fname)

try:

out = open(fname, 'w')

out.write(xml\_header)

out.write("<Vulscan file=\"%s\" time=\"%s\">\n" % (access, curtime))

for attack\_type in flag:

name = ""

if attack\_type in names:

name = " name=\"%s\"" % names[attack\_type]

out.write(" <attack type=\"%s\"%s>\n" % (attack\_type, name))

impacts = flag[attack\_type].keys()

impacts.sort(reverse=True)

for i in impacts:

out.write(" <impact value=\"%d\">\n" % int(i))

for e in flag[attack\_type][i]:

out.write(" <item>\n")

out.write(" <reason><![CDATA[%s]]></reason>\n" % e[2])

out.write(" <line><![CDATA[%s]]></line>\n" % e[3])

out.write(" </item>\n")

out.write(" </impact>\n")

out.write(" </attack>\n")

out.write("</Vulscan>")

out.close()

except IOError:

print "Cannot open the file:", fname

return

return

def generate\_html\_file(flag, access, filters, odir, vr):

curtime = time.strftime("%a-%d-%b-%Y", time.localtime())

fname = '%s\_Vulscan\_%s.html' % (access, curtime)

fname = os.path.abspath(odir + os.sep + fname)

try:

out = open(fname, 'w')

out.write(html\_header)

out.write('<div class="part1-content">')

out.write('<h1 class = "title" >General Vulnerability Meter </h1> <div class = "subtitle">[%s]</div>\n' % (curtime))

out.write(" <div class='progress-bar vr' data-percent='%d' data-duration='1000' data-color='#bdc3c7,#e74c3c' ></div>" % (int(vr)\*100))

if vr > 0 :

out.write("<div class ='section'>")

out.write("<div class = 'attack'>Tempering was observed in the system</div>")

if not filecmp.cmp('/etc/hosts', './template\_files/hosts\_temp'):

out.write("<div class='block highlight'>")

out.write("<div class='reason' style='font-style:normal; margin-top:20px; padding-top: 20px'><div class='headers'>Host Configuration</div>")

out.write("Host Configuration was seen to be tempered in this system.")

out.write("</div></div>")

if not filecmp.cmp('/etc/nsswitch.conf', './template\_files/nsswitch\_temp'):

out.write("<div class='block highlight'>")

out.write("<div class='line' style='padding-top: 20px'><div class='headers'>Network Service Switching</div>")

out.write("Service switching and networking configuration is tempered in this system")

out.write("</div></div>")

if not filecmp.cmp('/etc/resolv.conf', './template\_files/resolv\_temp'):

out.write("<div class='block highlight'>")

out.write("<div class='regexp' style='padding-top: 20px'><div class='headers'>Resolve</div>")

out.write("Dynamic resolution system of the server is tempered.")

out.write("</div></div>")

out.write("</div>")

out.write('</div>')

out.write('<div class="part2-content">')

out.write('<h1 class = "title" >VulScan Report </h1> <div class = "subtitle">[%s]</div>\n' % (curtime))

for attack\_type in flag:

print attack\_type

for attack\_type in flag:

name = ""

if attack\_type in names:

name = "%s" % names[attack\_type]

if len(flag[attack\_type].values()) < 1:

continue

else:

print names[attack\_type]

out.write("<div class = 'section'><div class='attack'>%s <span class='fullname'>(%s)</span></div>\n" % (attack\_type, name))

impacts = flag[attack\_type].keys()

impacts.sort(reverse=True)

for i in impacts:

out.write("<div class='match impact-%d'>\n" % int(i))

out.write(" <div class='progress-bar' data-percent='%d' data-duration='1000' data-color='#bdc3c7,#e74c3c' ></div>" % (int(i)\*10))

for e in flag[attack\_type][i]:

out.write(" <div class='block highlight'>\n")

out.write(" <div class='reason'><div class='headers'>Reason:</div> %s</div>\n" % html\_entities(e[2]))

out.write(" <div class='line'><div class='headers'>Log line:</div>%s</div>\n" % html\_entities(e[0][5]))

out.write(" <div class='regexp'><div class='headers'>Matching Signature:</div>%s</div>\n" % html\_entities(e[1]))

out.write(" </div>\n")

out.write("</div>\n")

out.write("<br />\n")

out.write("</div>\n")

out.write("</div>\n")

out.close()

except IOError:

print "Cannot open the file:", fname

return

months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

def correct\_period(date, period):

date = date.replace(':', '/')

l\_date = date.split('/')

for i in (2,1,0,3,4,5):

if i != 1:

cur = int(l\_date[i])

if cur < period['start'][i] or cur > period['end'][i]:

return False

else:

cur = months.index(l\_date[i])

if cur == -1:

return False

if cur < period['start'][i] or cur > period['end'][i]:

return False

return True

def analyze\_date(date):

d\_min = [01, 00, 0000, 00, 00, 00]

d\_max = [31, 11, 9999, 24, 59, 59]

date = date.replace(':', '/')

l\_date = date.split(';')

l\_start= l\_date[0].split('/')

l\_end = l\_date[1].split('/')

v\_start = [01, 00, 0000, 00, 00, 00]

v\_end = [31, 11, 9999, 24, 59, 59]

for i in range(len(l\_start)):

if l\_start[i] == '\*': continue

else:

if i == 1:

v\_start[1] = months.index(l\_start[1])

else:

cur = int(l\_start[i])

if cur < d\_min[i]: v\_start[i] = d\_min[i]

elif cur > d\_max[i]: v\_start[i] = d\_max[i]

else: v\_start[i] = cur

for i in range(len(l\_end)):

if l\_end[i] == '\*': continue

else:

if i == 1:

v\_end[1] = months.index(l\_end[1])

else:

cur = int(l\_end[i])

if cur < d\_min[i]: v\_end[i] = d\_min[i]

elif cur > d\_max[i]: v\_end[i] = d\_max[i]

else: v\_end[i] = cur

return {'start' : v\_start, 'end' : v\_end}

def main(argc, argv):

filters = "default\_filter.xml"

access = "access\_log"

output = ""

preferences = {

'attack\_type' : [],

'period' : {

'start' : [01, 00, 0000, 00, 00, 00],# day, month, year, hour, minute, second

'end' : [31, 11, 9999, 24, 59, 59]

},

'except' : False,

'exhaustive' : False,

'encodings' : False,

'output' : "",

'odir' : os.path.abspath(os.curdir),

'sample' : float(100)

}

if argc < 2 or sys.argv[1] == "--help":

help()

sys.exit(0)

else:

for i in range(argc):

s = argv[i]

if i < argc:

if s in ("--filters","-f"):

filters = argv[i+1]

elif s in ("--log","-l"):

access = argv[i+1]

elif s in ("--output", "-o"):

preferences['odir'] = argv[i+1]

print preferences['odir']

elif s in ("--html", "-h"):

preferences['output'] += ",html"

print "cool"

elif s in ("--xml", "-x"):

preferences['output'] += ",xml"

elif s in ("--text", "-t"):

preferences['output'] += ",text"

else:

print "argument error, '%s' has been ignored" % s

if len(preferences['output']) < 1:

preferences['output'] = "text"

if not os.path.isdir(preferences['odir']):

print "The directory %s doesn't exist, Vulscan will try to create it"

try:

os.mkdir(preferences['odir'])

except:

print "/!\ Vulscan cannot write in",preferences['odir']

print "/!\ Ising /tmp/Vulscan/ as new directory..."

preferences['odir'] = '/tmp/Vulscan'

os.mkdir(preferences['odir'])

Vulscaner(access, filters, preferences)

if \_\_name\_\_ == "\_\_main\_\_":

main(len(sys.argv), sys.argv)

Snapshots:

Fig. Start scan



Fig. HTML Report for General Vulnerability



Fig HTML Reports for Specific Vulnerabilities Fig 1

Fig 2



Fig 3



XML Report generated by vulscanner



Text Report Generated by vulscanner

