Lab 13: Analysis Automation Tools

- VirtualBox
- VMWare
- TShark

Automated Malware Analysis VirtualBox

Initial VirtualBoxSetup (previous lab)

- Automation in Python
- Run vmauto.py: specifically designed for automating malware analysis.
- Use vmauto.py for your automated script
- See malwarecookbook/8/vmauto.py (in the tool folder included in the document)

vmauto.py:

#!/usr/bin/python

 # Copyright (C) 2010 Michael Ligh

#

 # This program is free software: you can redistribute it and/or modify

 # it under the terms of the GNU General Public License as published by

 # the Free Software Foundation, either version 3 of the License, or

 # (at your option) any later version.

#

 # This program is distributed in the hope that it will be useful,

 # but WITHOUT ANY WARRANTY; without even the implied warranty of

 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

 # GNU General Public License for more details.

#

 # You should have received a copy of the GNU General Public License

 # along with this program. If not, see <http://www.gnu.org/licenses/>.

#

```
# [NOTES] ---------------------------------------
# 1) If you're running VirtualBox on Windows, you'll need win32com, which is
#    included in the Python Extensions for Windows package
#-----------------------------------------------
import sys, os, time, glob, shutil
from optparse import OptionParser
import subprocess

#-----------------------------------------------

vm_paths = {
   # the standard path on Mac OS X
   '/Library/Application Support/VMware Fusion/vmrun': 'fusion',
   # the standard path on Linux
   '/usr/bin/vmrun': 'ws',
   # the standard path on Windows
   'C:\\Program Files\\VMware\\VMware Workstation\\vmrun.exe': 'ws',
   }

def pinfo(msg):
   print "[INFO] ", msg

def perror(msg):
   print "[ERROR] ", msg

#-----------------------------------------------

class VBoxAuto:
   def __init__(self, machine):
```

```
    self.machine = machine
    self.ctx   = {}
    self.mach   = None

def get_mach(self):
    return  self.ctx['global'].getArray(self.ctx['vb'],  'machines')

def check(self):

    try:
        from vboxapi import VirtualBoxManager
    except ImportError:
        perror('You need to install VirtualBox!')
        return False

    vbm = VirtualBoxManager(None, None)

    self.ctx = {'global':vbm,
                'const' :vbm.constants,
                'vb'   :vbm.vbox,
                'mgr'   :vbm.mgr}

    # the machine name or id must be valid

    for m in self.get_mach():
        if m.name == self.machine or m.id == self.machine:
            self.mach = m
            break
```

```
    if self.mach == None:
        perror('Cannot find the machine: %s' % self.machine)
        return False

    pinfo('Using %s (uuid: %s)' % (self.mach.name, self.mach.id))

    pinfo('Session state: %s' % self.get_const(
        "SessionState", self.mach.sessionState))
    pinfo('Machine state: %s' % self.get_const(
        "MachineState", self.mach.state))

    return True

def get_const(self, enum, elem):
    # this lookup fails on Python2.6 - if that happens
    # then just return the element number
    try:
        all = self.ctx['const'].all_values(enum)
        for e in all.keys():
            if str(elem) == str(all[e]):
                return e
    except:
        return '%d' % elem

def list(self):
    try:
        for m in self.get_mach():
            print "%-12s %s (state:%s/%s)" %(m.name, m.id,
                self.get_const("MachineState", m.state),
```

```
            self.get_const("SessionState", m.sessionState))
    except:
        perror('No machines. Did you call check() first?')

def start(self, nsecwait=20):
    vb      = self.ctx['vb']
    session = self.ctx['mgr'].getSessionObject(vb)
    p = vb.openRemoteSession(session, self.mach.id, 'gui', '')

    while not p.completed:
        p.waitForCompletion(1000)
        self.ctx['global'].waitForEvents(0)

    if int(p.resultCode) == 0:
        session.close()
    else:
       perror('Cannot start machine!')

    pinfo('Waiting %d seconds to boot...' % nsecwait)
    time.sleep(nsecwait)

def opensession(self):
    session = self.ctx['global'].openMachineSession(self.mach.id)
    mach = session.machine
    return (session, mach)

def closesession(self, session):
    self.ctx['global'].closeMachineSession(session)
    time.sleep(5)
```

```
def stop(self):
    (session, mach) = self.opensession()
    pinfo('Powering down the system')
    try:
        session.console.powerDown()
        time.sleep(5)
        self.closesession(session)
    except Exception, e:
        pinfo(e)

def revert(self, snapname):
    # Revert a VM to the specified snapshot
    (session, mach) = self.opensession()
    pinfo("Reverting to snapshot '%s'" % snapname)
    try:
        snap = mach.findSnapshot(snapname)
        session.console.restoreSnapshot(snap)
        time.sleep(5)
        self.closesession(session)
    except Exception, e:
        pinfo(e)

def winexec(self, user, passwd, args):
    (session, mach) = self.opensession()
    try:
        argstr = ' '.join(args[1:])
    except:
        argstr = ''
```

```
        pinfo("Executing '%s' with args '%s'" % (args[0], argstr))
        pinfo("If this set fails, set up autologin for your user.")
        env = []
        ret = session.console.guest.executeProcess(
           args[0],
           0,
           args,
           env,
           user, passwd, 0)
        # on Windows, executeProcess returns an IProgress instance
        if os.name == "nt":
           pid = ret[3]
        else:
           pid = ret[1]
        pinfo('Process ID: %d' % pid)

# --------------------------------------------

 class VMwareAuto:
    def __init__(self, vmx):
        self.vmx = vmx

        self.vmrun  = None
        self.vmtype = None

        if not os.path.isfile(vmx):
           raise 'Cannot find vmx file in ' + vmx

        for (path,type) in vm_paths.items():
```

```
        if os.path.isfile(path):
            self.vmrun = path
            self.vmtype = type
            break

    if self.vmrun == None:
        raise 'Cannot find vmrun in ' + ','.join(vm_paths.keys())
    else:
        print 'Found vmrun (running on %s)' % self.vmtype

def setuser(self, user, passwd):
    '''
    Sets the credentials on the guest machine to
    use when copying files to/from the guest and
    when executing programs in the guest
    '''
    self.user = user
    self.passwd = passwd

def run_cmd(self, cmd, args=[], guest=False):
    '''
    Execute a command through vmrun. Additional
    parameters for commands can be set with args[]
    '''
    print 'Executing ' + cmd + ' please wait...'
    pargs = [self.vmrun, '-T', self.vmtype]
    if guest:
        pargs.extend(['-gu', self.user])
        pargs.extend(['-gp', self.passwd])
```

```
    pargs.append(cmd)
    pargs.append(self.vmx)
    pargs.extend(args)

    proc = subprocess.Popen(
        pargs,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT
    )
    return proc.communicate()[0]

def list(self):
    '''
    List the running virtual machines
    '''
    pargs = [self.vmrun, 'list']
    print pargs
    proc = subprocess.Popen(
        pargs,
        stdout=subprocess.PIPE
    )
    return proc.communicate()[0]

def start(self):
    '''
    Start the virtual machine specified by self.vmx
    '''
    return self.run_cmd('start')
```

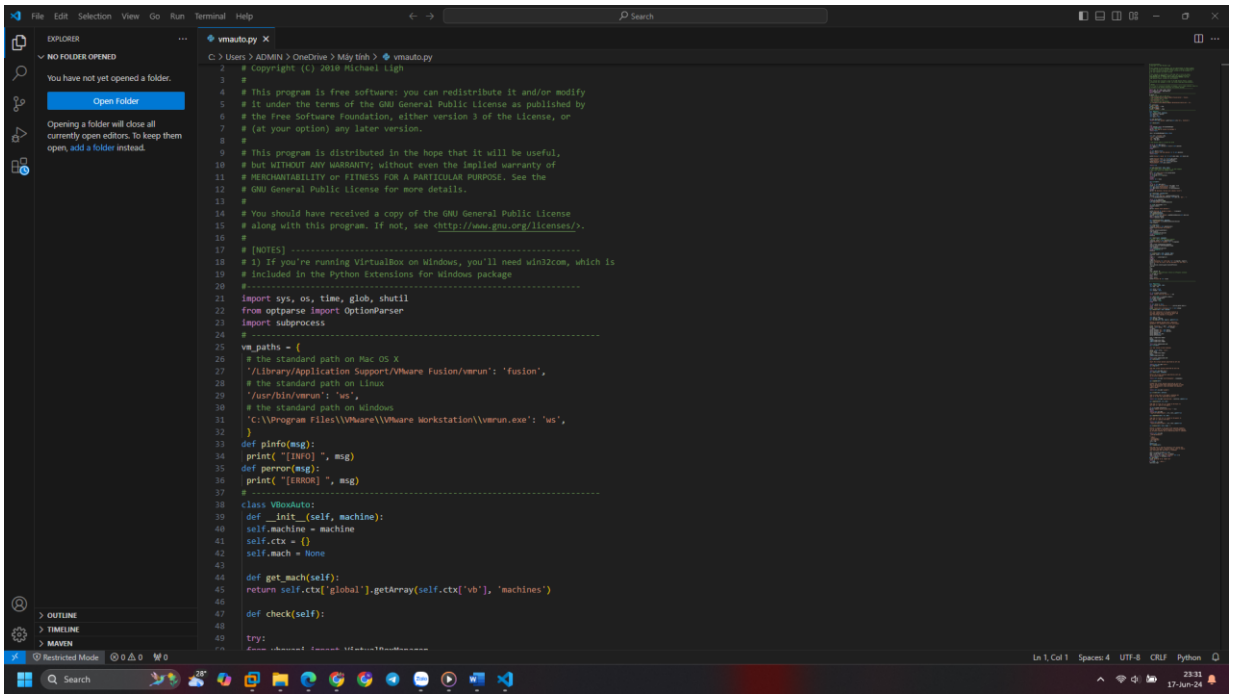
```
def stop(self):
    '''
    Stop the virtual machine specified by self.vmx
    '''
    return self.run_cmd('stop')

def revert(self, snapname):
    '''
    Revert the virtual machine specified by self.vmx
    to the given snapshot
    '''
    return self.run_cmd('revertToSnapshot', [snapname])

def suspend(self):
    '''
    Suspend the virtual machine specified by self.vmx.
    This is usually done after executing malware in order
    freeze the machine's state and obtain its physical
    memory sample
    '''
    return self.run_cmd('suspend')

def scrshot(self, outfile):
    '''
    Take a screen shot of the guest's desktop and
    save it to the file specified by outfile
    '''
    return self.run_cmd('captureScreen', [outfile], guest=True)
```

```
def copytovm(self, src, dst):
    '''
    Copy the src file (src is a path on the host) to
    dst (dst is a path on the guest).
    '''
    if not os.path.isfile(src):
        perror('Cannot locate source file ' + src)
        return
    return self.run_cmd(
        'copyFileFromHostToGuest', [src, dst], guest=True)


def copytohost(self, src, dst):
    '''
    Copy the src file (src is a path on the guest) to
    dst (dst is a path on the host).
    '''
    return self.run_cmd(
        'copyFileFromGuestToHost', [src, dst], guest=True)


def winexec(self, file, args=''):
    '''
    Execute a command in the guest with supplied arguments.
    You can use this to execute malware or existing programs
    on the guest machine such as monitoring tools or whatever.
    '''
    return self.run_cmd(
        'runProgramInGuest',
        [
            '-noWait',
```

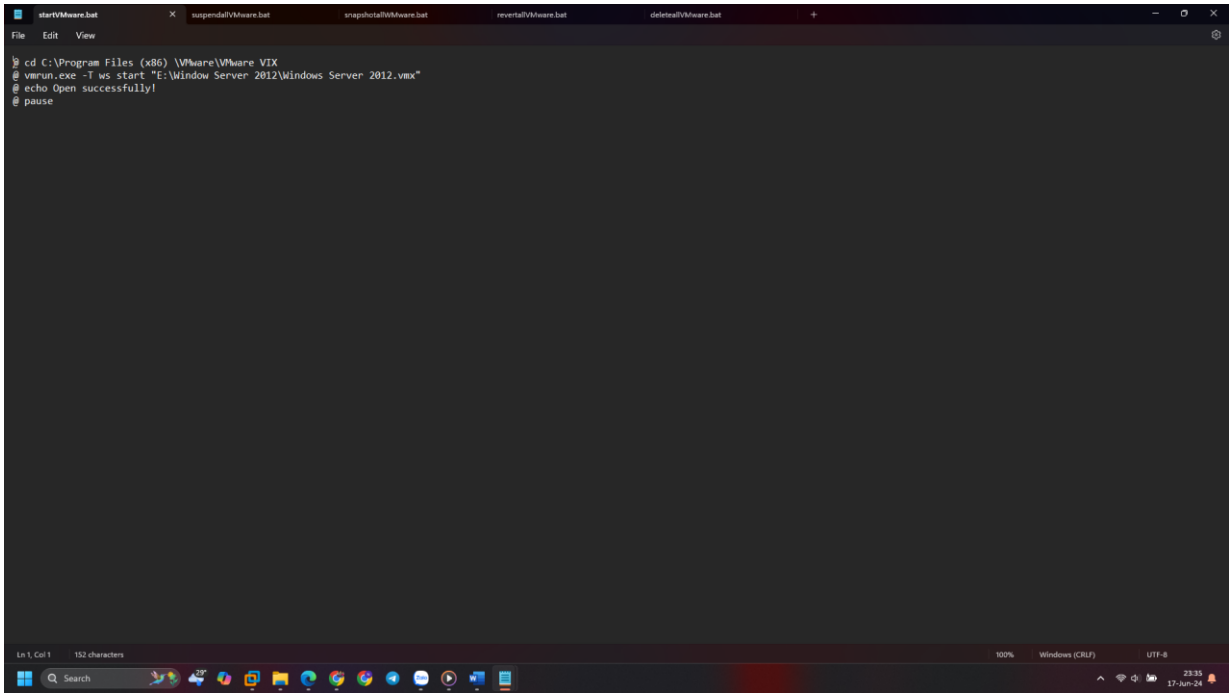
```
            '-interactive',
            '-activeWindow',
            file, args
        ],
        guest=True)

    def findmem(self):
        '''
        Find the file on the host machine's file system that
        represents the guest's physical memory. This is usually
        only available when the guest is suspended
        '''
        path = os.path.dirname(self.vmx)
        mems = glob.glob('%s/*.vmem' % (path))
        mems = [m for m in mems if "Snapshot" not in m]
        return mems[0] if len(mems) else ''

def main(argv):
    print 'Nothing to do. Import me!'
    return 0

if __name___== '__main__':
    main(sys.argv)
```

Automated Malware Analysis VMWARE

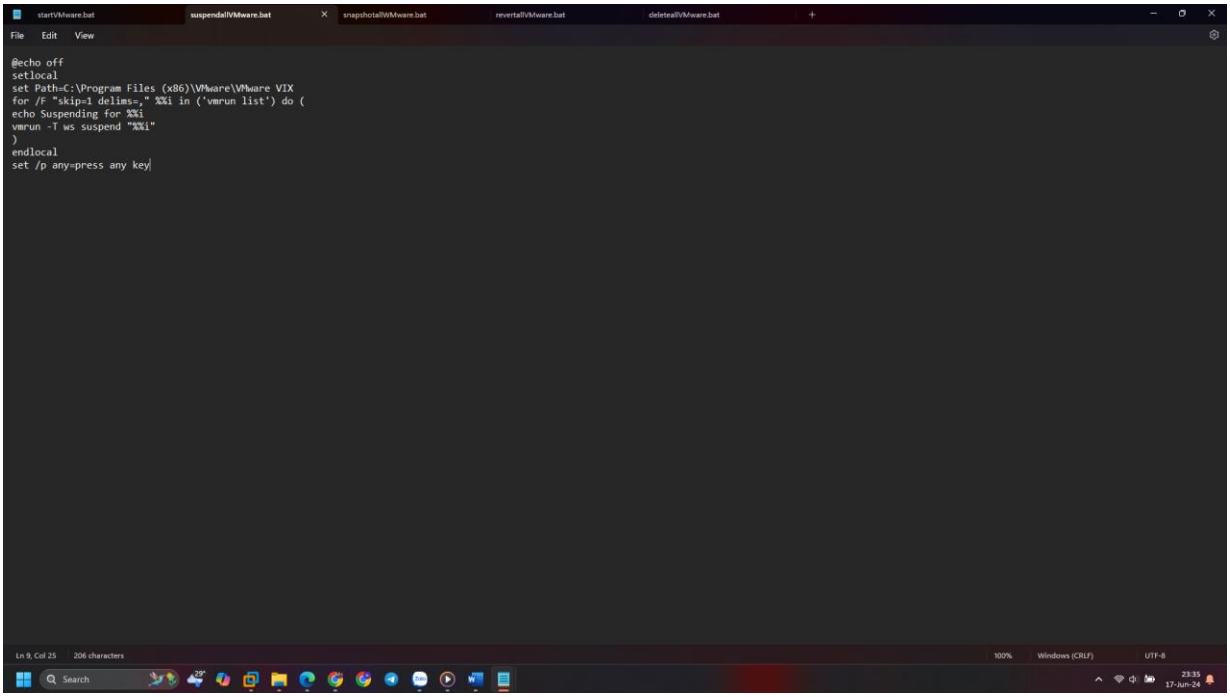
Steps: Create script to turn on Windows Sever 2012 virtual machine in Vmware.


File Edit Selection View Go Run Terminal Help
Search
EXPLORER
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently open editors. To keep them open, add a folder instead.
vmauto.py
Users > ADMIN > OneDrive > Máy tính > vmauto.py
Copyright (C) 2010 Michael Ligh
#
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
#
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
#
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.
#
[NOTES] ---
1) If you're running VirtualBox on Windows, you'll need win32com, which is
included in the Python Extensions for Windows package
#---
import sys, os, time, glob, shutil
from optparse import OptionParser
import subprocess
# --
vm_paths = {
# the standard path on Mac OS X
'/Library/Application Support/VMware Fusion/vmrun': 'fusion',
the standard path on Linux
'/usr/bin/vmrun': 'ws',
the standard path on Windows
'C:\\Program Files\\VMware\\VMware Workstation\\vmrun.exe': 'ws',
}
def pinfo(msg):
print("[INFO] ", msg)
def perror(msg):
print("[ERROR] ", msg)
# --
class VMonAuto:
def __init__(self, machine):
self.machine = machine
self.ctx = {}
self.mach = None
def get_mach(self):
return self.ctx['global'].getArray(self.ctx['vb'], 'machines')
def check(self):
try:
OUTLINE
TIMELINE
MAVEN
Restricted Mode
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
Python
Search
23:31
17-Jun-24

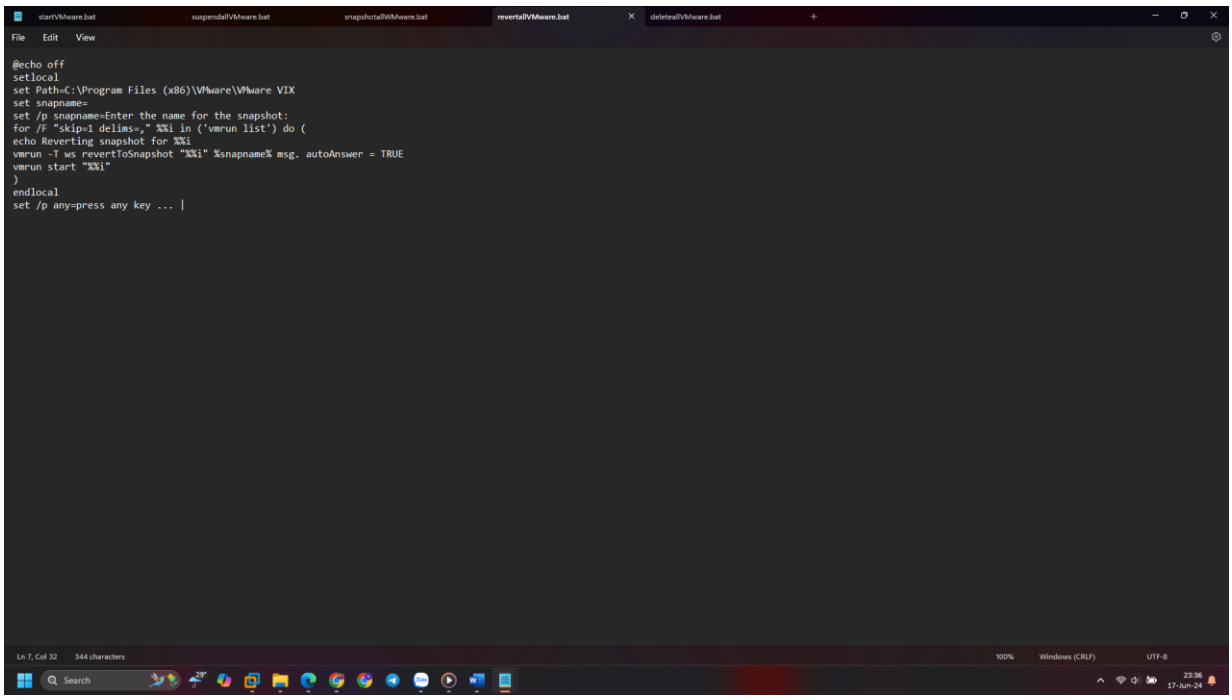
Create a script to suspend all virtual machines running on VMware.



Create a script to create snapshots for virtual machines running on VMware.



Create a script to revert all snapshots for virtual machines running on VMware.



Create a script to delete newly created snapshot files from the above script