VolatilityBot

Installation and plugin development guide

Martin G. Korman

Nov. 2015

System installation

Introduction

Download the Source code from bitbucket, at the following address:
https://bitbucket.org/martink90/volatilitybot_public
Unzip it, and copy it to the location where you want to install VolatilityBot.

Preparing the host

VolatilityBot was tested in Mac OS Yosemite and on various Ubuntu distributions. General requirements are as follows:

1. Volatility 2.5, and all its pre-requisites documented in the project site.
2. Python 2.7
3. Python Libraries:
   3.1. distorm3
   3.2. pefile
   3.3. yara
   3.4. sqlalchemy
   3.5. pydasm
   3.6. yaml
4. VMware workstation for Linux, VMware Fusion for Mac.

Creating and configuring the Virtual Machines

Create a new windows virtual machine , choose From WinXPx86 to Win7x64. Windows 8 or Windows 10 machines were not tested, but should work if Volatility Supports. After installation was complete do the following:

1. Disable Windows FW
2. Disable Windows updates
3. Install python 2.7
4. Add the following script to the startup folder, and name it agent.pyw (Note *.pyw extension and not *.py):
5. Change the IP address in s.bind to the IP of the virtual machine.

You can change the address to 0.0.0.0 to listen at any address.

```
import socket
import sys
import subprocess

s = socket.socket()
s.bind(("192.168.47.20",9999))
s.listen(1) # Acepta hasta 10 conexiones entrantes.
sc, address = s.accept()

print address
i=1
f = open('c:\\binary_to_exec.exe','wb')
i=i+1


l = sc.recv(1024)
while (l):
    f.write(l)
    l = sc.recv(1024)
f.close()


sc.close()
s.close()
```

6. Launch the script, and any additional programs you want to keep open, and take a VM snapshot.
7. Clone as many machines as you like, remember to change the IP in the script.
8. The machine pools features enables you to send the same sample to multiple different Virtual Machines. You can create additional machines with different operating system as well. Just repeat these steps.

Creating the configuration files

The main configuration file is called main.conf and can be found at conf/ folder:

```
mainconfig:
 general:
  VolatilityBot_Home: /home/martin/MWA/VolatilityBot/volatilitybot
  machine_type: vmware
  db_engine: "sqlite:////home/martin/MWA/db.sqlite3"
  volatility_path: "/usr/local/bin/vol.py"
  modules: malfind,modscan,procdump
  active_pools: WinXPx86,Win7x64
  log_path : /home/martin/MWA/VolatilityBot/volatilitybot/logs
  machine_failure_threshold : 6
```

Change the configuration as follows:

* VolatilityBot_Home - This is the path you copied VolatilityBot to
* machine_type - Leave it as vmware
* db_engine - This is the path to your sqlite DB, which you will create later. This file can get really big.
* volatility_path - The path where vol.py is stored.
* modules - Choose the Code extractor plugins you want to be used every execution:
    * malfind, modscan, procdump, apihooks
* active_pools - Which machine pools will be used. Give a name to each machine pool you created in the previous section - use a descriptive name (If you created only one, put only that one).
* log_path - Where the execution log will be stored
* machine_failure_threshold - How much times a VM has to fail to revert/shutdown before it is set as failed, and not used for the rest of the analysis queue.

Inside the configuration folder, there is a file name yara_rules.yar. Save inside it all the YARA rules you want to.

There is an additional configuration file inside conf/ named vmware.conf:
The VMware configuration file has a JSON structure as follows:

General VMware configuration:

* vmrun_path - A path where VMware's vmrun binary is stored, which is used by VolatilityBot to manage the virtual machines
* machine_resource_pools - Each section has to have the same name as configured in main.conf

Each of the resource pool sections has the following parameters:

* active - Tell wether this resource pool is active or not
* profile - The Volatility profile used for this machine type. The list can be seen using the vol.py command:

```
[martin@VolatilityBot-Core-0 conf]$ vol.py --info
Volatility Foundation Volatility Framework 2.4


Profiles
--------
VistaSP0x64     - A Profile for Windows Vista SP0 x64
VistaSP0x86     - A Profile for Windows Vista SP0 x86
VistaSP1x64     - A Profile for Windows Vista SP1 x64
VistaSP1x86     - A Profile for Windows Vista SP1 x86
VistaSP2x64     - A Profile for Windows Vista SP2 x64
VistaSP2x86     - A Profile for Windows Vista SP2 x86
Win2003SP0x86   - A Profile for Windows 2003 SP0 x86
Win2003SP1x64   - A Profile for Windows 2003 SP1 x64
Win2003SP1x86   - A Profile for Windows 2003 SP1 x86
Win2003SP2x64   - A Profile for Windows 2003 SP2 x64
Win2003SP2x86   - A Profile for Windows 2003 SP2 x86
Win2008R2SP0x64 - A Profile for Windows 2008 R2 SP0 x64
Win2008R2SP1x64 - A Profile for Windows 2008 R2 SP1 x64
Win2008SP1x64   - A Profile for Windows 2008 SP1 x64
Win2008SP1x86   - A Profile for Windows 2008 SP1 x86
Win2008SP2x64   - A Profile for Windows 2008 SP2 x64
Win2008SP2x86   - A Profile for Windows 2008 SP2 x86
Win2012R2x64    - A Profile for Windows Server 2012 R2 x64
Win2012x64      - A Profile for Windows Server 2012 x64
Win7SP0x64      - A Profile for Windows 7 SP0 x64
Win7SP0x86      - A Profile for Windows 7 SP0 x86
Win7SP1x64      - A Profile for Windows 7 SP1 x64
Win7SP1x86      - A Profile for Windows 7 SP1 x86
Win8SP0x64      - A Profile for Windows 8 x64
Win8SP0x86      - A Profile for Windows 8 x86
Win8SP1x64      - A Profile for Windows 8.1 x64
Win8SP1x86      - A Profile for Windows 8.1 x86
WinXPSP1x64     - A Profile for Windows XP SP1 x64
WinXPSP2x64     - A Profile for Windows XP SP2 x64
WinXPSP2x86     - A Profile for Windows XP SP2 x86
WinXPSP3x86     - A Profile for Windows XP SP3 x86
```

```
{
    "general": {
        "vmrun_path": "/usr/bin/vmrun",
        "machine_resource_pools": {
            "WinXPx86": {
                "active": true,
                "profile": "WinXPSP3x86",
                "machines": {
                    "wxp_01": {
                        "status": "idle",
                        "vmdk_path": "/home/martin/vmware/wxp_01/",
                        "name": "wxp_01",
                        "IP": "192.168.47.20",
                        "enabled": true,
                        "snapshot_name": "VolatilityBot",
                        "vmx_path": "/home/martin/vmware/wxp_01/wxp_01.vmx",
                        "pool": "WinXPx86"
                    },
                    "wxp_02": {
```

* machines - this section has a sub-section for each virtual machine configured:
  * status - Leave as idle
  * vmdk_path - The path where the VMware disk file is stored
  * name - The name of the virtual machine, as it appears in VMware
  * IP - The address of the VM

* enabled - true/false

* snapshot_name - The name of the snapshot you took before, with the agent.pyw running.

* vmx_path - The path where the VMware config file for the machine is stored.

* pool - The resource pool this machine belongs to

It is recommended to validate the JSON structure with JSlint after saving this file.

Initialise the DB, and create the Golden Images

Inside the Utils folder, there is a script name db_builder.py. Execute it in order to create the DB file.

In the same folder, there is a script named gi_builder.py. Execute it and will launch each virtual machine configured, and save a golden image of the configured virtual machines. Each machine is reverted and after 10 seconds suspended and all relevant data is saved. This steps takes around 25 seconds per VM.

Test VolatilityBot

1. The main script is VolatilityBot.py:

```
usage: VolatilityBot.py [-h] [-f FILENAME] [-s SLEEP] [-r] [-x] [-e] [-D]
[-S]
                        [-Q] [-t TAGS]

optional arguments:
  -h, --help            show this help message and exit
  -f FILENAME, --filename FILENAME
                        The Executable you want to submit
  -s SLEEP, --sleep SLEEP
                        How much time to wait, in seconds
  -r
  -x                    Delete original file after submission
  -e                    Enqueue files, but do not analyze them now
  -D                    Execute in daemon mode
  -S                    Skip existing samples
  -Q                    Reprocess samples that were in waiting state and
                        failed samples
  -t TAGS, --tags TAGS  Sample tags, separated by commas i.e: Dyre,Upatre
```

2. In order to test you installed VolatilityBot correctly, Launch it in Daemon mode:

```
[martin@VolatilityBot VolatilityBot]$ python VolatilityBot.py -D --sleep 120
```

3. In a separate terminal, submit a sample:

```
[martin@VolatilityBot VolatilityBot]$ python VolatilityBot.py -e –f /tmp/sh.exe --sleep 120
```

4. If you have configured everything correctly, you should see now that VolatilityBot is reverting virtual machines and after 2 minutes suspending them and extracting all malicious code found.
5. After VolatilityBot has finished, you will found all related output files in the Store folder.

YARA Semantic analyzer (YSA) and writing rules for it

YARA Semantic analyzer is a post-processing module, that uses sophisticated YARA rules in order to find various behaviour patterns in the malicious code extracted from memory. As default the only code extractor configured to use YSA is Malfind.

This is an example of a YSA rule file with some included rules:

```
{
    "yara_rules": [
        {
            "pattern": "68 [4] FF 15 API:LoadLibraryA 50 FF 15 API:GetProcAddress",
            "rule_name": "generic_dynamic_lib_loading"
        },
        {
            "pattern": "FF 15 API:CreateToolhelp32Snapshot [-] 50 57 FF 15
API:Process32FirstW [-] FF 15 API:Process32NextW",
            "rule_name": "generic_process_iterator"
        },
        {
            "pattern": "6A 06 6A 01 6A 02 FF 15 API:WSASocketW [-] API:WSAConnect",
            "rule_name": "socket_tcp"
        },
        {
            "pattern": "6A 17 6A 02 6A 01 FF 15 API:WSASocketW [-] API:WSAConnect",
            "rule_name": "socket_udp"
        },
        {
            "pattern": "FF 15 API:GetCurrentProcess [-] 50 FF 15 API:OpenProcessToken [-]
68 string:SeDebugPrivilege [-] FF 15 API:LookupPrivilegeValueW [-]
API:AdjustTokenPrivileges",
            "rule_name": "Process_Privilege_Escalation"
        }
    ]
}
```

Let me explain the rule socket_tcp:

This rule allows VolatilityBot to detect socket creation patterns in the extracted code.

```
6A 06 – push 0x6 protocol (6 means TCP)
6A 01 – push 0x1 type (TCP type)
6A 02 – push 0x2 af (IPv4)
FF 15 API:WSASocketW – FF 15 is the call opcode for a function, and on runtime – YSA will
resolve the offset of WSASocketW according to the fixed binary (With fixed IAT) that came as
output the code extractor (malfind in this case)
```

```
[-] - wildcard (YARA Standard)
API:WSAConnect - A call to the function that established the connection
```

Plugin Development Guide

There are two types of plugins that can be created:

1. Code Extractors
2. Post Processing modules

Code Extractors

The core plugins of VolatilityBot, Without them there is no extraction of malicious code from memory. Let's take a relatively simple code extractor, and dissect it in order to understand how to write one:

The code extractor needs a configuration reading function, That will load all required parameters from the configuration file and/or additional configuration files if required. (i.e Your Code Extractor plugin needs one)

```
def _load_config():
  global VolatilityBot_Home
  global volatility_path

  if os.path.isfile('conf/main.conf'):
      f = open('conf/main.conf')
      # use safe_load instead load
      dataMap = yaml.safe_load(f)
      f.close()

      volatility_path  = pipes.quote(dataMap['mainconfig']['general']
['volatility_path'])

      return True
```

This function will be called at the beginning of the main function of the code extractor, which has to be named _run, and receive the following parameters:

```
def _run(vm_name,f_profile,vmem_path,workdir,sample_id):
```

These parameters are passed by the manager, every time the plugin is called:

vm_name - The name of the machine on which the malware was executed

f_profile - The Volatility profile of that machine

vmem_path - The path to the memory dump

workdir - The directory to which the output has to be stored

sample_id - The ID of the sample, given by the manager

This is the initialisation sequence inside the _run function of the mod scan code extractor, which extracts loaded kernel modules from memory:

```
1  global VolatilityBot_Home
2  global volatility_path
3  _load_config()
4  mod_white_list = ['kmixer.sys','Bthidbus.sys']
5
6
7
8  #Get golden image data
9  modscan_gi = []
10 f = open(VolatilityBot_Home + '/GoldenImage/' + vm_name + '/modscan', 'r')
11 modscan_GoldenImage = f.readlines()
12
13
14
```

Lines 1-2: Declaration of global variables

Line 3: A small whitelist, of common legitimate loading drivers. (BT and Sound card)

Lines 4-6: Loading a golden image from disk to modscan_GoldenImage variable.

Golden Image is the output of the code extractor module when executed on a specific machine in clean state. The golden image is used in order to avoid false positives.

Creation of a golden image:

In the utils folder, there is a script named gi_builder.py. This script is used as part of the installation, or when updating your machines or creating new ones. Each machine must have a golden image for each code extractor that requires one. This is the part of code in gi_builder that is responsible of saving the golden images (This code is execute once per machine):

```
            modules = ['pslist','dlllist','ldrmodules','modscan']
            for mod in modules:
                command = volatility_path + ' --profile ' + vm['profile'] + ' -f ' +
vmem_path + ' ' + mod
                print command
                proc = subprocess.Popen(command, shell=True,stdout=subprocess.PIPE)
                print '[*] Executing ' + mod
                output = proc.stdout.read()

                #Write output to file
                obj = open(gi_dir + '/' + mod, 'wb')
                obj.write(output)
                obj.close
```

If your plugin is based on a simple Volatility command, you can just add it to the list inside modules. If not, just add the code required for your golden image creation right after this loop (Outside of this loop).

Note that since **Volatility 2.5**, the —output=json parameter can be used, and parsing plugins output is much easier.

Post Processing Modules

These modules are called by the code extractors in the following convention (This code is taken from malfind code extractor, which calls a YARA post processing module):

```
        #yara output:
        yara_output = yara_postprocessor._run(outputpath,sample_id)
        if (yara_output != "none"):
              obj = open(outputpath + '.yara_results', 'wb')
              obj.write(yara_output)
              obj.close
```

The post processing module main function must be named _run, and it will receive the following parameters:

outputpath - The path of the extracted code located in the store on disk.

sample_id - The ID of the sample given by the manager

It is recommended that the plugin will output a JSON, alt ought it is not obligatory. This is an example of the main function in the YARA post processing module:

```
def _run(filename,f_sample_id):
    _init()
    global sample_id
    sample_id = f_sample_id
    yarfile = VolatilityBot_Home + '/conf/yara_rules.yar'
    print "[*] Loaded .yar file: %s" % (yarfile)
    rules = yara.compile(yarfile)
    matches = rules.match(filename,callback=showdata)
    json_output =  json.dumps(results_list, indent=4, sort_keys=True)
    if (rules_matched):
        return json_output
    else:
        return "none"
```

Line 1: An internal initialisation function

Line 6: Compilation of the YARA rule file

Line 7: Execution of the YARA rules on the input file.

Line 9: The JSON is returned by the function