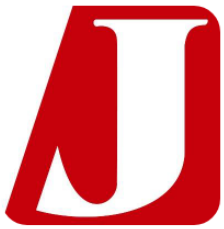
aa-tools/cobaltstrikescan.py at master · JPCERTCC/aa-tools · GitHub

github.com/JPCERTCC/aa-tools/blob/master/cobaltstrikescan.py

JPCERTCC

JPCERTCC/**aa-tools**



Artifact analysis tools by JPCERT/CC Analysis Center

11 Contributors | 2 Issues | 410 Stars | 90 Forks

```
# Detecting CobaltStrike for Volatility
#
# How to use:
# 1. locate "cobaltstrikescan.py" in [Volatility_Plugins_Directory]
# ex) mv cobaltstrikescan.py /usr/lib/python2.7/dist-packages/volatility/plugins/malware
# 2. python vol.py [cobaltstrikescan | cobaltstrikeconfig ] -f images.mem --profile=Win7SP1x64

import volatility.plugins.taskmods as taskmods
import volatility.win32.tasks as tasks
import volatility.utils as utils
import volatility.debug as debug
import volatility.plugins.malware.malfind as malfind
```

```
import volatility.plugins.malware as malware
import re
from struct import pack, unpack, unpack_from, calcsize
from socket import inet_ntoa

try:
import yara
has_yara = True
except ImportError:
has_yara = False

cobaltstrike_sig = {
'namespace1' : 'rule CobaltStrike { \
strings: \
$v1 = { 73 70 72 6E 67 00} \
$v2 = { 69 69 69 69 69 69 69 69} \
condition: $v1 and $v2}'
}

CONF_PATTERNS = [{
'pattern': '\x69\x68\x69\x68\x69',
'cfg_size': 0x1000,
'cfg_info': [['\x00\x01\x00\x01\x00\x02', 'BeaconType\t\t:', 0x2],
['\x00\x02\x00\x01\x00\x02', 'Port\t\t\t:', 0x2], ['\x00\x03\x00\x02\x00\x04',
'Polling(ms)\t\t:', 0x4],
['\x00\x04\x00\x02\x00\x04', 'Unknown1\t\t:', 0x4], ['\x00\x05\x00\x01\x00\x02',
'Jitter\t\t\t:', 0x2], ['\x00\x06\x00\x01\x00\x02', 'Maxdns\t\t\t:', 0x2],
['\x00\x07\x00\x03\x01\x00', 'Unknown2\t\t:', 0x100], ['\x00\x08\x00\x03\x01\x00',
'C2Server\t\t:', 0x100], ['\x00\x09\x00\x03\x00\x80', 'UserAgent\t\t:', 0x80],
```

```
['\x00\x0a\x00\x03\x00\x40', 'HTTP_Method2_Path\t:', 0x40],
['\x00\x0b\x00\x03\x01\x00', 'Unknown3\t\t:', 0x100], ['\x00\x0c\x00\x03\x01\x00',
'Header1\t\t\t:', 0x100],

['\x00\x0d\x00\x03\x01\x00', 'Header2\t\t\t:', 0x100], ['\x00\x0e\x00\x03\x00\x40',
'Injection_Process\t:', 0x40], ['\x00\x0f\x00\x03\x00\x80', 'PipeName\t\t:', 0x80],

['\x00\x10\x00\x01\x00\x02', 'Year\t\t\t:', 0x2], ['\x00\x11\x00\x01\x00\x02', 'Month\t\t\t:',
0x2], ['\x00\x12\x00\x01\x00\x02', 'Day\t\t\t:', 0x2],

['\x00\x13\x00\x02\x00\x04', 'DNS_idle\t\t:', 0x4], ['\x00\x14\x00\x02\x00\x04',
'DNS_sleep(ms)\t\t:', 0x2], ['\x00\x1a\x00\x03\x00\x10', 'Method1\t\t\t:', 0x10],

['\x00\x1b\x00\x03\x00\x10', 'Method2\t\t\t:', 0x10], ['\x00\x1c\x00\x02\x00\x04',
'Unknown4\t\t:', 0x4], ['\x00\x1d\x00\x03\x00\x40', 'Spawnto_x86\t\t:', 0x40],

['\x00\x1e\x00\x03\x00\x40', 'Spawnto_x64\t\t:', 0x40], ['\x00\x1f\x00\x01\x00\x02',
'Unknown5\t\t:', 0x2], ['\x00\x20\x00\x03\x00\x80', 'Proxy_HostName\t\t:', 0x80],

['\x00\x21\x00\x03\x00\x40', 'Proxy_UserName\t\t:', 0x40], ['\x00\x22\x00\x03\x00\x40',
'Proxy_Password\t\t:', 0x40], ['\x00\x23\x00\x01\x00\x02', 'Proxy_AccessType\t:', 0x2],

['\x00\x24\x00\x01\x00\x02', 'create_remote_thread\t:', 0x2]]

}]


BEACONTYPE = {0x0: "0 (HTTP)", 0x1: "1 (Hybrid HTTP and DNS)", 0x8: "8 (HTTPS)"}

ACCESSTYPE = {0x1: "1 (use direct connection)", 0x2: "2 (use IE settings)", 0x4: "4
(use proxy server)"}



class cobaltstrikeScan(taskmods.DllList):

    """Detect processes infected with CobaltStrike malware"""

    @staticmethod
    def is_valid_profile(profile):
        return (profile.metadata.get('os', 'unknown') == 'windows')

    def get_vad_base(self, task, address):
        for vad in task.VadRoot.traverse():
            if address >= vad.Start and address < vad.End:
```

```
return vad.Start

return None

def calculate(self):

if not has_yara:
debug.error("Yara must be installed for this plugin")

addr_space = utils.load_as(self._config)

os = self.is_valid_profile(addr_space.profile)
if not os:
debug.error("This command does not support the selected profile.")

rules = yara.compile(sources=cobaltstrike_sig)

for task in self.filter_tasks(tasks.pslist(addr_space)):
scanner = malfind.VadYaraScanner(task=task, rules=rules)

for hit, address in scanner.scan():

vad_base_addr = self.get_vad_base(task, address)

yield task, vad_base_addr
break

def render_text(self, outfd, data):

self.table_header(outfd, [("Name", "20"),
("PID", "8"),
("Data VA", "[addrpad]")])
```

```
for task, start in data:
self.table_row(
outfd, task.ImageFileName, task.UniqueProcessId, start)


class cobaltstrikeConfig(cobaltstrikeScan):

"""Parse the CobaltStrike configuration"""

def get_vad_end(self, task, address):
for vad in task.VadRoot.traverse():
if address == vad.Start:
return vad.End + 1

return None

def decode_config(self, cfg_blob):
return "".join(chr(ord(cfg_offset) ^ 0x69) for cfg_offset in cfg_blob)

def parse_config(self, cfg_blob, nw, outfd):
outfd.write("[CobaltStrike Config Info]\n")

for pattern, name, size in nw['cfg_info']:
if name.count('Port'):
port = unpack_from('>H', cfg_blob, 0xE)[0]
outfd.write("%s %d\n" % (name, port))
continue
offset = cfg_blob.find(pattern)
if offset == -1:
outfd.write("%s\n" % name)
continue
```

```
    config_data = cfg_blob[offset + 6:offset + 6 + size]
    if name.count('Unknown'):
    outfd.write("%s %s\n" % (name, repr(config_data)))
    continue
    if size == 2:
    if name.count('BeaconType'):
    outfd.write("%s %s\n" %
    (name, BEACONTYPE[unpack('>H', config_data)[0]]))
    elif name.count('AccessType'):
    outfd.write(
    "%s %s\n" % (name, ACCESSTYPE[unpack('>H', config_data)[0]]))
    elif name.count('create_remote_thread'):
    if unpack('>H', config_data)[0] != 0:
    outfd.write("%s Enable\n" % name)
    else:
    outfd.write("%s Disable\n" % name)
    else:
    outfd.write(
    "%s %d\n" % (name, unpack('>H', config_data)[0]))
    elif size == 4:
    if name.count('DNS_idle'):
    outfd.write("%s %s\n" % (name, inet_ntoa(config_data)))
    else:
    outfd.write("%s %d\n" %
    (name, unpack('>I', config_data)[0]))
    else:
```

```
    if name.count('Header'):
    outfd.write("%s " % name)
    cfg_offset = 3
    flag = 0
    while 1:
    if cfg_offset > 255:
    break
    else:
    if config_data[cfg_offset] != '\x00':
    if config_data[cfg_offset + 1] != '\x00':
    if flag:
    outfd.write("\t\t\t: ")
    outfd.write("%s\n" % config_data[
    (cfg_offset + 1):].split('\x00')[0])
    cfg_offset = config_data[cfg_offset:].find(
    '\x00\x00\x00') + cfg_offset - 1
    flag += 1
    else:
    cfg_offset += 4
    continue
    else:
    cfg_offset += 4
    continue
    else:
    outfd.write("%s %s\n" % (name, config_data))

    def render_text(self, outfd, data):
```

```
delim = '-' * 70

for task, start in data:
proc_addr_space = task.get_process_address_space()

data = proc_addr_space.zread(
start, self.get_vad_end(task, start) - start)

for nw in CONF_PATTERNS:
cfg_addr = data.find(nw['pattern'])
if cfg_addr != -1:
break
else:
continue

outfd.write("config addr: %08X\n\n" % cfg_addr)
outfd.write("{0}\n".format(delim))
cfg_blob = data[cfg_addr:cfg_addr + nw['cfg_size']]
outfd.write("Process: %s (%d)\n\n" %
(task.ImageFileName, task.UniqueProcessId))
self.parse_config(self.decode_config(cfg_blob), nw, outfd)
```