Malware Analysis With Multiple Features

Muhammad Najmi Ahmad Zabidi

UKSIM 2012
Emmanuel College
University of Cambridge, United Kingdom

28-30th March 2012
Disclaimer

This presentation is as extension of the previous industry talks that I presented in Hack In The Box 2011, Kuala Lumpur. Some contents are based from the previous talk.
Malware in short

- is a software
- maliciousness is defined on the risks exposed to the user
- sometimes, when in vague, the term “Potentially Unwanted Program/Application” (PUP/PUA) being used
Methods of detections

- Static analysis
- Dynamic analysis
This talk is more *static analysis*
Analysis of strings

- Important, although *not foolproof*
- Find *interesting calls* first
- Considered *static analysis*, since no executing of the binary
Methods to find interesting strings

- Use `strings` command (on *NIX systems)
- Editors
- Checking with Import Address Table (IAT)
Python

- a scripting language
- a robust, powerful programming language
My Python scripts

- Based from several existing Python scripts - malware analyzer, zerowine sandboxes, PE scanner
- I merged them and modified some parts so that it will be able to produce single page of report
- This tool is needed for my research work (bigger objective)
- Analysis of the binary while it is still packed
Components of pi-ngaaji

- Anti VM detector
- API calls detector
- Anti debugger detector
- Other Interesting calls
- DLL detector
- Binary entropy calculator
- XOR decryptor module
Stuffs to look at

- “Interesting” Application Programming Interface-API calls
- Virtual Machine(VM) detector
- Outbound connect, especially Internet Relay Chat-IRC commands. Possibly a member of botnets
- XOR’ed values
python-pefile module

- Written by Ero Carrera
- python-pe provides quite a number of functions
- Everything can be dumped by `print pe.dump_info()`
import re provides regexp capability to find strings in the binary. This array of calls INTERESTING_CALLS = ["CreateMutex"...], provides ranges of calls to be fetched. The following fetched the represented strings:

```python
for calls in INTERESTING_CALLS:
    if re.search(calls, line):
        if not calls in performed:
            print "[+] Found an Interesting call to: ", calls
            performed.append(calls)
```
API calls

- Application Programming Interface - API calls
- We use and compare the original API calls embedded in the script by Joxean, and later use the API calls proposed by [Altaher et al., 2011]
Some DLLs are interesting to look at, they contain functions that me be used for malicious activities. For e.g: Kernel32.dll, provides “low-level operating system functions for memory management and resource handling”
Contents of kernel32.dll

1. CopyFileA
2. CopyFileExA
3. CopyFileExW
4. CopyFileW
5. CreateFileA
6. CreateFileW
7. DeleteFileA
8. DeleteFileW
9. MoveFileA
10. MoveFileExA
11. MoveFileExW
12. MoveFileW
13. MoveFileWithProgressA
14. MoveFileWithProgressW
15. OpenFile
16. ReadFile
17. ReadFileEx
18. ReadFileScatter
19. ReplaceFile
20. ReplaceFileA
21. ReplaceFileW
22. WriteFile
23. WriteFileEx
24. WriteFileGather

Source: [Marhusin et al., 2008]
import hashlib
import time
import binascii
import string
import os, sys
import commands
import pefile
import peutils
import string

pe = pefile.PE(sys.argv[1])
print "DLL		API NAME"
for imp in pe.DIRECTORY_ENTRY_IMPORT:
    print imp.dll
for api in imp.imports:
    print "	\t%s" %api.name
najmi@vostro:~/rogue-av$ avgscan BestAntivirus2011.exe
AVG command line Anti-Virus scanner
Copyright (c) 2010 AVG Technologies CZ

Virus database version: 271.1.1/3943
Virus database release date: Fri, 07 Oct 2011 14:34:00 +08:00

BestAntivirus2011.exe Trojan horse FakeAlert.ACN

Files scanned : 1(1)
Infections found : 1(1)
PUPs found : 0
Files healed : 0
Warnings reported : 0
Errors reported : 0

najmi@vostro:~/rogue-av$ md5sum BestAntivirus2011.exe
7f0ba3e7f57327563f0ceacbd08f8385  BestAntivirus2011.exe
$ python ../dll-scan.py BestAntivirus2011.exe

<table>
<thead>
<tr>
<th>DLL</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVAPI32.dll</td>
<td>None</td>
</tr>
<tr>
<td>USER32.dll</td>
<td>None</td>
</tr>
<tr>
<td>KERNEL32.dll</td>
<td>None</td>
</tr>
<tr>
<td>ole32.dll</td>
<td>None</td>
</tr>
<tr>
<td>OLEAUT32.dll</td>
<td>None</td>
</tr>
<tr>
<td>GDI32.dll</td>
<td>None</td>
</tr>
<tr>
<td>COMCTL32.dll</td>
<td>None</td>
</tr>
<tr>
<td>SHELL32.dll</td>
<td>None</td>
</tr>
<tr>
<td>WININET.dll</td>
<td>None</td>
</tr>
<tr>
<td>WSOCK32.dll</td>
<td>None</td>
</tr>
</tbody>
</table>
Anti Virtual Machine Malware

"Red Pill": "\x0f\x01\x0d\x00\x00\x00\x00\xc3",
"VirtualPc trick": "\x0f\x3f\x07\x0b",
"VMware trick": "VMXh",
"VMCheck.dll": "\x45\xC7\x00\x01",
"VMCheck.dll for VirtualPC": "\x0f\x3f\x07\x0b\xc7\x45\xfc\xff\xff\xff\xff",
"Xen": "XenVMM", # Or XenVMMXenVMM
"Bochs & QEmu CPUID Trick": "\x44\x4d\x63",
"Torpig VMM Trick": "\xE8\xED\xFF\xFF\xFF\x25\x00\x00\x00\xFF
\x33\xC9\x3D\x00\x00\x00\x80\x0F\x95\xC1\x88\xC1\xC3",
"Torpig (UPX) VMM Trick": "\x51\x51\x0F\x01\x27\x00\xC1\xFB\xB5\xD5\x35
\x02\xE2\xC3\xD1\x66\x25\x32
\xBD\x83\x7F\xB7\x4E\x3D\x06\x80\x0F\x95\xC1\x88\xC1\xC3"

Source: ZeroWine source code
Strings detector

```
INTERESTING_CALLS = ['CreateMutex', 'CopyFile', 'CreateFile', 'WRITE', 'NtCreateFile', 'call shell32', 'advapi32.RegOpenKey',
                     'Kernel32.CreateProcess', 'shdocvw', 'gethostname', 'ws2_32.bind', 'ws2_32.listen', 'ws2_32.htons',
                     'advapi32.RegCreate', 'advapi32.RegSet', 'http://', 'Socket',
                     '"\{\?\}d\d\d\d2[0-4]d[25][0-5]\}.\{\?\}d\d\d\d2[0-4]d[25][0-5]\}.\{\?\}d\d\d\d2[0-4]d[25][0-5]\}2[0-4]d[25][0-5]\}',
                     'OutputDebugString', 'GetEnvironmentStrings', 'LoadLibraryA', 'WSASocketA', 'GetProcAddress',
                     'FindWindow', 'CreateProcess', 'DuplicateTokenEx', 'ImpersonateNamedPipeClient', 'RevertToSelf', 'signal',
                     'IsDebuggerPresent']

INTERESTING_CALLS_DLLS=["KERNEL32.DLL", "advapi32.dll", "comctl32.dll", "gdi32.dll", "ole32.dll", "oleaut32.dll", "user32.dll", "wscket32.dll", "ntdll.dll"]

INTERESTING_SYS_CALLS=["ping.exe", "telnet.exe"]

REGISTRY_CALLS = ['HKEY_CURRENT_USER', 'HKEY_CLASSES_ROOT', 'HKEY_LOCAL_MACHINE', "autorun.inf"]

ONLINE_WORK = ['IRC', 'Joined channel', 'Port', 'BOT', 'Login', 'Flood', 'ddos', 'NICK', 'ECHO', 'PRIVMSG', 'ADMIN',
               'KICK', 'CONNECT', 'LIST', 'MODE', 'MOTD', 'PING', 'PONG', 'QUIT', 'SERVLIST', 'SERVICE', 'NAMES', 'JOIN', 'INVITE', 'INFO', 'TRACE', 'USERHOST', 'WHO', 'WHOIS', 'VERSION']
```
Detect Anti VMs

```python
$python comp-detect.py vm-detect-malware/bfe00ca2aa27501cb4fd0065543555d

DLL     API NAME
WS2_32.dll CoCreateInstance
KERNEL32.dll
USER32.dll
GDI32.dll
ole32.dll

[+] Detecting Anti Debugger Tricks...
*** Detected trick TWX (TRW detection)
*** Detected trick isDebuggerPresent (Generic debugger detection)
*** Detected trick TRW (TRW detection)

[+] Detecting VM tricks..
*** Detected trick VirtualPc trick
*** Detected trick VMCheck.dll for VirtualPC

Analyzing registry...
Check whether this binary is a bot...
Analyzing interesting calls..
[+] Found an Interesting call to: CreateMutex
[+] Found an Interesting call to: GetEnvironmentStrings
[+] Found an Interesting call to: LoadLibraryA
[+] Found an Interesting call to: GetProcAddress
[+] Found an Interesting call to: IsDebuggerPresent
```
Detect Bots, Detect Debugger Detector

Analyzing 013a6dd86261acc7f9907740375ad9da DLL
API NAME
KERNEL32.dll
USER32.dll
ADVAPI32.dll
MSVCRT.dll
GDI32.dll
ole32.dll
SHELL32.dll
Detection Icon

Detecting VM existence...

No trick detected.
Analyzing registry...
Check whether this binary is a bot...
[+] Malware Seems to be IRC BOT: Verified By String: Port
[+] Malware Seems to be IRC BOT: Verified By String: SERVICE
[+] Malware Seems to be IRC BOT: Verified By String: Login

Analyzing interesting calls..
[+] Found an Interesting call to: LoadLibraryA
[+] Found an Interesting call to: GetProcAddress
[+] Found an Interesting call to: IsDebuggerPresent
[+] Found an Interesting call to: http://
Analyzing e65297bf9dbb2b2790e4d898d70c9e9

Analyzing registry...
[+] Malware is Adding a Key at Hive: HKEY_LOCAL_MACHINE
  \g@Labell1\^@^\A^\AŔ^\Nreg add "$HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\Rx.exe" /v debugger /t REG_SZ /d %systemroot%\repair\lsass.exe /f

....

[+] Malware Seems to be IRC BOT: Verified By String : ADMIN
[+] Malware Seems to be IRC BOT: Verified By String : LIST
[+] Malware Seems to be IRC BOT: Verified By String : QUIT
[+] Malware Seems to be IRC BOT: Verified By String : VERSION

Analyzing interesting calls...
[+] Found an Interesting call to: FindWindow
[+] Found an Interesting call to: LoadLibraryA
[+] Found an Interesting call to: CreateProcess
[+] Found an Interesting call to: GetProcAddress
[+] Found an Interesting call to: CopyFile
[+] Found an Interesting call to: shdocvw
Checking entropy

- Looking at randomness in the binary
- Entropy - referring to Shannon’s entropy [Lyda and Hamrock, 2007]
- If the score is $X>0$ and $X<1$ or $X>7$, it is being denoted as *suspicious*
- python-pefile modules provides `get_entropy()` function for this
PE sections to look for

TEXT
DATA
.idata
.rdata
.reloc
.rsrc
.tls
Binary file structure

Figure: Structure of a file [Pietrek, 1994]
Figure: PE components, simplified
print "\n[+]Now check for binary entropy.."

for sec in pe.sections:
    #s = "%-10s %-12s %-12s %-12s %-12f" % (s = "%-10s %-12s" % ('''.join([c for c in sec.Name if c in string.printable]), sec.get_entropy())
    if sec.SizeOfRawData == 0 or (sec.get_entropy() > 0 and sec.get_entropy() < 1) or sec.get_entropy() > 7:
        s += "[SUSPICIOUS]"

print "",s
Checking entropy. . .

[+] Now check for binary entropy..
%s .text     6.84045277182
%s .rdata    0.0          [SUSPICIOUS]
%s .data     7.99566735324 [SUSPICIOUS]
%s .ice      6.26849761461
Figure: pi-ngaji flow
## Results

**Table: API calls detection for pi-ngaji**

<table>
<thead>
<tr>
<th>API calls</th>
<th>Number of hits over 23 samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSystemTimeAsFileTime</td>
<td>1</td>
</tr>
<tr>
<td>SetUnhandledExceptionFilter</td>
<td>1</td>
</tr>
<tr>
<td>GetCurrentProcess</td>
<td>3</td>
</tr>
<tr>
<td>TerminateProcess</td>
<td>1</td>
</tr>
<tr>
<td>LoadLibraryExW</td>
<td>0</td>
</tr>
<tr>
<td>GetVersionExW</td>
<td>0</td>
</tr>
<tr>
<td>GetModuleFileNameW</td>
<td>0</td>
</tr>
<tr>
<td>GetTickCount</td>
<td>2</td>
</tr>
<tr>
<td>SetLastError</td>
<td>2</td>
</tr>
<tr>
<td>GetCurrentProcessId</td>
<td>2</td>
</tr>
<tr>
<td>GetModuleHandleW</td>
<td>2</td>
</tr>
<tr>
<td>LoadLibraryW</td>
<td>0</td>
</tr>
<tr>
<td>InterlockedExchange</td>
<td>1</td>
</tr>
<tr>
<td>UnhandledExceptionFilter</td>
<td>2</td>
</tr>
<tr>
<td>FreeLibrary</td>
<td>6</td>
</tr>
<tr>
<td>GetCurrentThreadId</td>
<td>3</td>
</tr>
<tr>
<td>QueryPerformanceCounter</td>
<td>1</td>
</tr>
<tr>
<td>CreateFileW</td>
<td>0</td>
</tr>
<tr>
<td>InterlockedCompareExchange</td>
<td>0</td>
</tr>
<tr>
<td>UnmapViewOfFile</td>
<td>0</td>
</tr>
<tr>
<td>GetProcAddress</td>
<td>12</td>
</tr>
<tr>
<td>VM/Debugger Tricks</td>
<td>Number of hits</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>RedPill</td>
<td>1</td>
</tr>
<tr>
<td>VMCheck</td>
<td>2</td>
</tr>
<tr>
<td>VMWare trick</td>
<td>1</td>
</tr>
<tr>
<td>IsDebuggerPresent</td>
<td>1</td>
</tr>
<tr>
<td>TRW</td>
<td>4</td>
</tr>
<tr>
<td>TRX</td>
<td>3</td>
</tr>
</tbody>
</table>
pi-ngaji strengths

- Works offline, no need to submit to honeypot/dynamic analysis *yet*
- Could be automated and generate reports - via UNIX pipe for e.g
- Runs on relatively secure environment - *Linux* - where win32 could not possibly execute
Cons

Weakness

- Could not possibly handle obfuscated binaries... too bad you have to execute it to get all the API/activities
Email me if you have questions

najmi.zabidi @ gmail.com
Special thanks

