Malware Analysis With Multiple Features

Muhammad Najmi Ahmad Zabidi
Kulliyyah of Information and Communication Technology,
International Islamic University Malaysia,
53100 Gombak,
Kuala Lumpur,
Malaysia
najmi@kict.iium.edu.my

Mohd Aizaini Maarof† and Anazida Zainal‡
Information Assurance and Security Research Group,
Faculty of Computer Science and Information Systems,
Universiti Teknologi Malaysia,
81310 Skudai, Johor Bahru,
Malaysia
aizaini@utm.my†, anazida@utm.my‡

Abstract—Malware analysis process is being categorized into static analysis and dynamic analysis. Both static and dynamic analysis have their own strengths and weaknesses. In this paper, we present a tool written in Python programming language called as pi-ngaji, which could assist the work of malware analyst to get the static features of malware. pi-ngaji contains several modules - Application Programming Interface (API) calls extractor, binary entropy information, anti virtual machine and anti debugger detector and XOR encrypted strings decryptor. pi-ngaji was developed in order to assist our work in getting malware features. pi-ngaji is focusing on ripping Microsoft Windows executable binaries’ malicious features.

Keywords-malware, static analysis, feature selection

I. INTRODUCTION

Malware is a threat to every single computer user, regardless of their level of usage and skill proficiency. Recent news as in [1], showed the danger of a malware, where a convicted murder got a retrial after his court transcript was deleted by a computer virus.

In the malware detection process, there are two analysis types; those are static and dynamic analysis.

Static analysis, according to [2] includes “the process of evaluating an executable based on its form, structure, and content, without actually executing the program”. Simply said, static analysis is where the malware researchers analyzed the malware without actually executing them. One of the strengths of static analysis is that it reduces the risk of harming the analyst’s computing host.

Dynamic analysis, in the other hands, is where the malware analysts execute the malware in a safe, emulated environment which emulates the real computing environment so that the real activities and behaviors of the malware can be monitored.

Malware could contains readable strings, in which by using text parser we could be able to get them. In the event where the strings are encrypted, they could be in the XOR format which is a lightweight method of encryption, which suits with “fast and small” malware for binary execution. pi-ngaji largely borrows the code from Zero Wine[3], an open source malware sandbox which was written by Joxean Koret. The other modules, for example XOR decryptor was originated from Rudnai[4] and Didier’s [5] idea. Initially, the large part of the code meant to work within the ZeroWine sandbox environment, but we need to make it work simply on the terminal. It depends heavily on Python PE package which was written by Ero Carrera. As of now we simply call the external program written by Rudnai[4] after few modifications.

A. The purpose of static analysis

Static analysis is a safe way to analyzed a malware, and for this purpose, pi-ngaji is tool to make the analysis faster. In certain conditions, static analysis provides useful informations faster than dynamic analysis which needs the malware to be upload or executed in the sandbox. Although with some limitations, it could at least analyze an unknown binary whether or not it falls into suspicious class of software or not because it does not depend on the malware signature, instead the malicious features that the malware has.

B. Contribution

For our research, we basically utilize pi-ngaji for feature parsing process, that is, a preprocess work that we need for feature selection. An example of utilizing such feature is explained in [6], which utilized Support Vector Machine (SVM) for detecting unknown virus. In this paper we combined several modules from several programs into a single package of program, which we named as pi-ngaji. Several alteration being done, for example we took the features that were proposed by [7] in order to evaluate the accuracy of those features. Some of the features by [7] actually are overlapped with the original features proposed by [3] hence we combined them altogether. We improved the original tool by [3] so that it will be able to work off line without booting into the virtual machine and we also embedded some other useful modules. We also added exclusive OR or XOR detector feature which was not in the original program.
II. RELATED WORKS

Author in [8] mentioned that the research of API calls detector that time was not able to detect virtual machine detector. In [9], the author focused on obfuscated calls, with a tool called as SAFE. Leder, et. al in [10] suggested that there is a need to analyze crypto function in a malware with a timely manner. For the API calls selected, [7] proposed the use of feature ranking by using Information Theory method. In our previous work [11], we proposed the use API calls as one of the features to be used in feature selection phase, in which the API calls being used was suggested to be reviewed so that the accuracy of detection could still be maintained while using minimal strong features. The details on how pi-ngaji works is described briefly in [12]. In [13] the author combined both static and dynamic analysis methods, however with using the debugger for the static analysis work.

III. METHODOLOGY

pi-ngaji works by parsing the following components:

- **Suspicious API calls**

- **Virtual machine (VM) detector**
  - Malware sandbox usually uses virtual machine to emulate normal computing environment. Some malware detects the existence of virtual machine in order to know whether it runs in a safe (fragile, normal machine) or not (emulated system, used by malware researcher). If the malware found out that it runs on a virtual machine it will either changed its behavior to a safe activity or not executing at all.

- **Debugger detector**
  - Same as in VM environment before, but for the debugger detector feature the malware will detects whether it is being statically analyzed through reverse engineering or debugging process. If the malware found out that it was analyzed by a debugger it will changed its behavior to normal, safe activity as well. Some of the details is covered in [14].

- **Other calls**
  - There are other interesting calls, such as `ping` command or `telnet` command which pi-ngaji interested in. Also, commands such as Internet Relay Chat (IRC) commands are included inside this program.

- **Binary entropy calculator**
  - Here, we put a threshold where X binary is considered suspicious when its value n is between 0 and 1 or more than 7. The value that we used was actually derived from the work written in [15], which suggested suspicious level of entropy is above 7. While [3] use entropy more than 0 and below than 1 as well as entropy more than 7, that is $(0 < z < 1 \lor z > 7)$. One of the recent papers up to this point of writing is [16] on several malware analysis using binary entropy.
  - Briefly, the structure of a portable executable file (here we call it PE to be simple), is as in Figure 1:

- **XOR encryption decryptor**
  - XOR means “exclusive OR” in which the operation is matching the assigned character with a key (in 0 and 1). If the key is match it will gives 0, if not will gives 1, that is:
    
    Given the character “A” in ASCII is 1000001, XOR with a key 1111111 is:

    
    $1000001 \oplus 1111111 = 0111110$

- **Submission to Virus Total (VT)**
  - Here the user needs to put in API key which can be gained after the user register themselves to VirusTotal web page. We try to make it simpler so that the user does not need to use the API key to get a binary info in the future. The API key can simply be inserted by hard-coded method inside the program’s file.

We use the term module since we segregate the work of each phase into independent functions, hence it should be “modular”. Most of the parts are being written in Python programming language, and we hope to be convert the XOR module from the original language in Perl into the current program body. The modules are also outlined in Figure 2.

The algorithm of pi-ngaji detection is shown in Algorithm 1, where the detection actually goes from top-bottom.

We also evaluated features listed by [7], in order to improve the detection rate of the tool. Some of the features
Algorithm 1 Pseudocode for pi-ngaji

\begin{algorithm}
\begin{algorithmic}
\State begin
\State $pi$-ngaji function calls
\State \hspace{1em} find suspicious API calls\Comment{line 5}
\State \hspace{2em} do if $(x)\text{string\_found}$ then print($x$)
\State \hspace{2em} end fi\Comment{line 6}
\State \hspace{1em} find anti VM features \Comment{line 8}
\State \hspace{2em} do if $(y)\text{string\_found}$ then print($y$)
\State \hspace{2em} end fi\Comment{line 10}
\State \hspace{1em} find anti debugger features \Comment{line 12}
\State \hspace{2em} do if $(z)\text{string\_found}$ then print($z$)
\State \hspace{2em} end fi\Comment{line 14}
\State \hspace{1em} find IRC calls, other interesting calls \Comment{line 16}
\State \hspace{2em} do if $(\theta)\text{string\_found}$ then print($\theta$)
\State \hspace{2em} end fi\Comment{line 18}
\State \hspace{1em} calculate binary entropy results \Comment{line 20}
\State \hspace{2em} $z := 0$
\State \hspace{2em} do if \{$z > 0\}$ \AND \{$z < 1\}$ \OR \{$z > 7\}$
\State \hspace{2em} then print($file\_is\_susicious$)
\State \hspace{2em} end fi\Comment{line 22}
\State \hspace{1em} find XOR crypto \Comment{line 26}
\State \hspace{2em} do if $(q)\text{string\_found}$ then print($q$)
\State \hspace{2em} end fi\Comment{line 28}
\State \hspace{1em} end \Comment{line 30}
\end{algorithmic}
\end{algorithm}

listed in [7] were added in order to evaluate its effectiveness in the detection process.

In order to parse the input, the flows are depicted in the Figure 3:

**IV. ANALYSIS**

We analyzed the following features, with 22 malware samples and one benign sample (altogether 23 samples).

For the API call analysis as in Table I, it shows some of the features are not even being called. None of the API calls as in Table I detected in the benign sample. This is due to the nature of the software, it may not be using the API that being used for the analysis. Some of the API calls that exist in malware could exist in benign software too, and in the other way around.:

Table I

<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>GetTickCount</td>
<td>2</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>
</tbody>
</table>

Figure 3. pi-ngaji function flows
in malware, the following detection feature showed in Table II. This means, some of the malware does not have virtual machine/debugger detection traits at all.

For entropy analysis, most malware (around 80 percent) shows the level of entropy above the threshold, that is above 7.0. Some of the malware did not show the randomness of packed data within their binaries.

### Table III
**Binary Entropy Analysis for pi-ngaji**

<table>
<thead>
<tr>
<th>Number of files hit</th>
<th>Number of files unhit</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>4 (where one file is benign)</td>
</tr>
</tbody>
</table>

#### A. XOR analysis

For the XOR analysis of the previous 23 samples, most of the strings could be fetched by using Python PE modules. Hence we selected a special sample that was announced by F-Secure and the details is covered in [17]. We analyzed a sample with MD5 hash `e9f89d406e32ca88c32ac22852c25841`, in which the AVG anti virus called it “Downloader.Agent2.ATEM”. We managed to rip out the XOR'ed HTTP strings which related to the certificate issuer being used by the malware. The suspicious host (by the time of this writing, was inaccessible) which was connected to the malware is also successfully being exposed. This malware, upon the time of this paper is written, considered new and was widely covered in Malaysia since the certificate issuer being manipulated by the malware is one of two certificate providers in Malaysia.

### V. CONCLUSION

Static analysis, as one of the malware analysis method assists in the detection of software maliciousness as well as the classification of malware families. Though current malware research is not depending solely on the static analysis, instead combining it together with the dynamic analysis; hence hybrid analysis, it still has room of improvements due to the nature of malware. Zero day malware usually needs static analysis method since the current detection method largely depends on the signature based scanning. *pi-ngaji* helps in the sense it should be able to assist the basic analysis of a binary for the malware researcher.

### VI. FUTURE WORKS

*pi-ngaji* currently was not written in a single program, as for the XOR decryptor module we are yet to convert it to the single Python program. Nevertheless, for now it served the purpose of the analysis. The strength of this tool is that it is publicly available, since the originated code is in GNU Public License, so we have to inherit the previous license too. Also, this tool could provide “one stop station” since we put every useful function into the tool so that the analysis time could be minimized. One of the weakness of depending much on Python PE (Portable Executable) module is that it does not be able to proceed for analysis if the binary was compiled or written in non standard PE compliance binary. We managed to found out several binaries that worked that way, but most of the malware binaries were written and compiled in a standard manner. We plan to overcome this weakness in our future works. The features in API calls could also be reduced by using feature selection method since currently there are more than 20 API calls attributes that being used in the program.

### ACKNOWLEDGMENT

The authors want to thank Universiti Teknologi Malaysia for providing a research space for the research. Also, thanks to the International Islamic University Malaysia for providing funding to one of the researchers under the Endowment Grant (Type A), EDW A11-151-0942. Finally, the Ministry of Higher Learning (MOHE) of Malaysia for providing supports throughout the research period.
REFERENCES


