International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 22 (2018) pp. 15793-15797

Malicious Process Detection using OSA on a DFXML Evidence File

[1] Shruti B. Yagnik, [2] Dr. Binod C. Agrawal

[1]Calorx Teachers' University, Computer Science and Engineering Department, Ahmedabad, Gujarat, India.

[2]Calorx Teachers' University, Computer Science and Engineering Department, Ahmedabad, Gujarat, India.

Abstract

Cyber-Crime by running malicious processes on workstation are becoming rampant in information industries. Malicious processes tend to be dangerous to sensitive information and must be detected as and when they are triggered. Various tools and techniques are available to detect such dangerous processes out of which, we have set our focus on the DFXML (Digital Forensics XML) language that is used to exchange structured Cyber-Forensic information. We have an evidence file in the DFXML format captured from a malicious computer system which contains all running processes metadata and other related information. We have applied the OSA (Optimal String Alignment) algorithm to detect malicious processes from collected evidences in DFXML format. This has lessened the manual work of the cyber forensic investigators to a greater extent of comparing all running processes along with valid white-hat processes, aiding in faster malicious processes detection in Cyber Forensic Investigations.

Keywords: Cyber-Crime, Cyber-Forensics, Evidences, Forensic Investigations, Digital Forensic, Digital Forensic XML, Malicious Process, Optimal String Alignment Algorithm

I. INTRODUCTION

Many processes with malicious intentions can reside in computer systems, can be triggered by any external activity done with a malicious intension or done un-intentionally. Each process may have its own malicious intention that could be performed in background. To accomplish its goal, these processes may use variety of ways to hide themselves. One of the ways is that, they use process names nearly same as the legitimate processes and run in the computer system which goes un-noticed and taken as a genuine activity going on. This paper enlightens the way of detecting such file names which appear to be genuine at a glance but are actually black hatted processes with malicious code. There are lots and lots of processes running on the workstation. Checking for each and every process, manually, whether the name is genuine will be a tedious task and kill a lot of time thereby completing the work of the malware and fulfilling the attackers desire. Also there are other important places where manual intervention of the investigator is needed than spending it here on the process [4]. That is why we are automating the process and saving the time for the Investigators to thwart attack attempts and safeguard the node as fast as possible.

II. DIGITAL FORENSIC XTENSIBLE MARKUP LANGUAGE (DFXML)

Digital Forensic Xtensible Markup Language or the DFXML is a standard structured way to exchange forensic information between forensic tools. It stores information of various digital forensics objects. It provides structure to store information of persistent and volatile data. It includes storage structure for Windows Registry, Running Processes, Event Information and File and directory information. These structures are specially designed to store digital forensics information. Hence, Digital forensic investigators can use that information to predict the nature of evidences and use to prove committed crimes. We focused on running processes to discover malicious processes. All information related to running processes are stored in ProcessInformation.dfxml file which includes the following fields for the running processes [1][2].

- Process Id
- Parent Process Id
- Process name
- Owner
- Source Path
- Start time
- User time
- Kernel time
- Elapsed time
- Running time and many more

Each field stores information of a process. For example, we have collected an evidence file as below.

```
<?xml version="1.0"?><processes><process><pid>0</pid><ppid>0</ppid><pname>System Idle
Process</pname><status>running</status><memory-usage>24576</memory-
usage></process><process><pid>4</pid><ppid>0</ppid><pname>System</pname><status>running
</status><memory-usage>2625536</memory-
usage></process><process><pid>284</pid><ppid>4</ppid><pname>smss.exe</pname><status>run
ning</status><memory-usage>667648</memory-
usage></process><process><pid>392</pid><ppid>376</ppid><pname>csrss.exe</pname><status>
running</status><memory-usage>3096576</memory-
usage></process><process><pid>452</pid><ppid>376</ppid><pname>wininit.exe</pname><statu
s>running</status><memory-usage>3051520</memory-
usage></process><process><pid>460</pid><ppid>444</ppid><pname>csrss.exe</pname><status>
running</status><memory-usage>31891456</memory-
usage></process><process><pid>500</pid><ppid>452</ppid><pname>services.exe</pname><stat
us>running</status><memory-usage>5971968</memory-
usage></process><process><pid>516</pid><ppid>452</ppid><pname>lsass.exe</pname><status>
running</status><memory-usage>7512064</memory-
usage></process><process><pid>524</pid><ppid>452</ppid><pname>lsm.exe</pname><status>ru
nning</status><memory-usage>2625536</memory-
usage></process><process><pid>632</pid><ppid>500</ppid><pname>svchost.exe</pname><statu
s>running</status><memory-usage>6184960</memory-
usage></process><process><pid>688</pid><ppid>500</ppid><pname>scvhost.exe</pname><statu
s>running</status><memory-usage>5812224
```

Fig 1. Collected evidence of processes

The above file gives us various intuition about processes. Our goal is to detect malicious processes that may be hidden between legitimate processes. It is evident that two processes svchost.exe and scvhost.exe looks similar to each other. There is a need to devise a technique that can help digital forensic investigators to discover such similar processes and automate the task. Of course, it must reduce the time needed for digital forensic investigation process.

III. OPTIMAL STRING ALIGNMENT(OSA) ALGORITHM

We have focused on an algorithm that finds similar strings with minor difference by calculating distance between two strings. Digital forensic investigators can observe the processes having lesser distance and could be able to distinguish between black hat and white hat processes easily.

In information theory and computer science, the Damerau–Levenshtein distance (named after Frederick J. Damerau and Vladimir I. Levenshtein [6][7][8]) is a string metric for measuring the edit distance between two sequences. Informally, the Damerau–Levenshtein distance between two words is the minimum number of operations (consisting of insertions, deletions or substitutions of a single character, or transposition of two adjacent characters) required to change one word into the other.

The Damerau–Levenshtein distance differs from the classical Levenshtein distance by including transpositions among its allowable operations in addition to the three classical single-character edit operations (insertions, deletions and substitutions).[9][7]

In his seminal paper,[10] Damerau stated that these four operations correspond to more than 80% of all human misspellings. Damerau's paper considered only misspellings that could be corrected with at most one edit operation. While the original motivation was to measure distance between human misspellings to improve applications such as spell checkers, Damerau–Levenshtein distance has also seen uses in biology to measure the variation between protein sequences.[11]

It can be computed using variation of Wagner–Fischer dynamic programming algorithm. It computes the total operations needed to make distinct string to be equal. It does not allow editing string multiple times. This algorithm uses Levenshtein distance that defines the minimum number of alterations needed to make two distinct string identical. For example, The distance between CA and ABC using optimal string alignment algorithm is 3 vide CA→A→AB→ABC.

It requires less overhead. Hence, It works best at runtime operations.[12]

```
let d[0..length(a), 0..length(b)] be a 2-d array of integers, dimensions length(a)+1, length(b)+1
// note that d is zero-indexed, while a and b are one-indexed.

for i := 0 to length(a) inclusive do
    d[i, 0] := i
for j := 0 to length(b) inclusive do
    d[0, j] := j

for i := 1 to length(a) inclusive do
    for j := 1 to length(b) inclusive do
        if a[i] = b[j] then
            cost := 0
        else
            cost := 1
        d[i, j] := minimum(d[i-1, j] + 1,     // deletion
                           d[i, j-1] + 1,     // insertion
                           d[i-1, j-1] + cost)  // substitution
        if i > 1 and j > 1 and a[i] = b[j-1] and a[i-1] = b[j] then
            d[i, j] := minimum(d[i, j],
                               d[i-2, j-2] + cost)  // transposition
return d[length(a), length(b)]
```

The difference from the algorithm for Levenshtein distance is the addition of one recurrence:

```
if i > 1 and j > 1 and a[i] = b[j-1] and a[i-1] = b[j] then
    d[i, j] := minimum(d[i, j],
                       d[i-2, j-2] + cost)  // transposition
```

Fig 2. Pseudo-code for OSA

Example: To make two distinct strings “barking” and “dark”, few conversions may needed to make it identical

1. Barking -> barkin (deletion of g)
2. barkin→ barki (deletion of n)
3. barki→ bark (deletion of i)
4. bark→ dark (substitution of b)

Hence, the Levenshtein distance between these two words strings is 4.[13]

IV. IMPLEMENTATION

We have implemented the OSA algorithm using R programming and used ProcessInformation.dfxml file as an input to the algorithm. Following String Distance Matrix can be generated from given input process names. Process names are likely to be same if distance between two processes is lesser. Distance 0 between two strings indicates identical process names that should not be considered by digital forensic investigators. Instead, Distance 1 should be of interest of digital forensic investigators. Because, there is a single character mismatch between given two strings that may disguise a legitimate process.

	python	pythan	system idle process	system	smss	csrss	wininit	csrss	services	lsass
python	0	1	16	5	6	6	7	6	8	6
pythan	1	0	17	5	6	6	7	6	8	6
system idle process	16	17	0	13	15	15	18	15	13	16
system	5	5	13	0	4	5	7	5	6	5
smss	6	6	15	4	0	2	7	2	6	2
csrss	6	6	15	5	2	0	7	0	6	2
wininit	7	7	18	7	7	7	0	7	7	7
csrss	6	6	15	5	2	0	7	0	6	2
services	8	8	13	6	6	6	7	6	0	7
lsass	6	6	16	5	2	2	7	2	7	0
lsm	6	6	17	4	3	4	7	4	8	3
svchost	5	6	16	6	5	5	6	5	6	6
svchost	5	6	16	6	5	5	6	5	6	6
svchost	5	6	16	6	5	5	6	5	6	6
svchost	5	6	16	6	5	5	6	5	6	6
svchost	5	6	16	6	5	5	6	5	6	6
stacsv64	8	8	17	6	6	6	8	6	7	6
winlogon	6	7	16	8	8	8	5	8	8	7
svchost	5	6	16	6	5	5	6	5	6	6
docklogin	7	8	17	9	9	8	8	8	9	8
svchost	5	6	16	6	5	5	6	5	6	6
svchost	5	6	16	6	5	5	6	5	6	6
wlanext	7	7	17	6	7	7	5	7	8	6

Fig 3 String Distance Matrix (SDM)

Fig.2 shows String Distance Matrix (SDM) that show distance between various strings. Lesser distance signifies best match with the targeted string whereas higher distance signifies unmatched string.

It is quite evident from mentioned steps that digital forensics investigators can directly focus on targeted information. This process may reduce efforts of investigators to analyze various evidences. There were 93 processes running on computer system out of which only 5 processes have been deduced for further analysis.

V. LIMITATION

The above system has certain limitations. Inspite of digging out the malicious processes from a large pool, there are chances of false-negatives involved in it. There may be only 5 processes deduced out of 99,9999 processes running on a computer but it may not be necessary that all those 5 processes are malicious. To check and clarify that, manual intervention of the investigator is needed. He is saved only from checking the remaining 999994 processes. For the detected 5 he has to put on manual research and clarify whether they are actually malicious or not [3][5].

VI. CONLUSION

To recapitulate, we can say that a lot of time of the investigator is saved when out of the so many listed processes in the background, he needs to check only for some of the files that tend to be malicious. This time saving feature makes this OSA algorithm suitable to use for malicious file detection from a DFXML formatted evidence. The investigator can directly remove a very larger subset from the universal set of processes to check for malicious intensions. Time saving is a very crucial factor and at times it can save the node when maliciously running process is discovered and stopped from running completely and ruining the sensitive data. Along with time saving, it also reduces much of the tedious work of the investigator and nullifies biased views or manually done

errors in their work. As for example, there may be a slight change in the process name which may go undetected manually. It also aids in giving automated, quick and error free decisions.

REFERENCES

[1] Esan P. Panchal "Extraction of Persistence and Volatile Forensics Evidences from Computer System"*International Journal of Computer Trends and Technology (IJCTT)*,V4(5):964-968 May Issue 2013 .ISSN 2231-2803.www.ijcttjournal.org. Published by Seventh Sense Research Group.

[2] Premal C. Patel "Aggregation of Digital Forensics Evidences"*International Journal of Computer Trends and Technology (IJCTT)*,V4(4):881-884 April Issue 2013 .ISSN 2231-2803.www.ijcttjournal.org. Published by Seventh Sense Research Group.

[3] Machine Learning, T.M. Mitchell, McGraw Hill, 1997.

[4] Shruti B. Yagnik, "Requirements to Build a System that Uses Machine Learning Based Approach for Analysis of Forensic Data"*International Journal of Computer Trends and Technology (IJCTT)*,V4(4):927-932 April Issue 2013 .ISSN 2231-2803.www.ijcttjournal.org. Published by Seventh Sense Research Group.

[5] "Clustering-Kmeans" https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html(Accessed on 15/04/17)

[6] Brill, Eric; Moore, Robert C. (2000). An Improved Error Model for Noisy Channel Spelling Correction(PDF). Proceedings of the 38th Annual Meeting on Association for Computational Linguistics. pp. 286-293. doi:10.3115/1075218.1075255. Archived from the original (PDF) on 2012-12-21.

[7] Bard, Gregory V. (2007), "Spelling-error tolerant, order-independent pass-phrases via the Damerau–Levenshtein string-edit distance metric", Proceedings of the Fifth Australasian Symposium on ACSW Frontiers : 2007, Ballarat, Australia, January 30 - February 2, 2007, Conferences in Research and Practice in Information Technology, 68, Darlinghurst, Australia: Australian Computer Society, Inc., pp. 117–124, ISBN 978-1-r920682-49-1

[8] Li; et al. (2006). Exploring distributional similarity based models for query spelling correction (PDF). Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. pp. 1025–1032. doi:10.3115/1220175.1220304. Archived from the original (PDF) on 2010-04-01.

[9] Levenshtein, Vladimir I. (February 1966), "Binary codes capable of correcting deletions, insertions, and reversals", Soviet Physics Doklady, 10 (8): 707–710

[10] Damerau, Fred J. (March 1964), "A technique for computer detection and correction of spelling errors", Communications of the ACM, ACM, 7 (3): 171–176, doi:10.1145/363958.363994

[11] The method used in: Majorek, Karolina A.; Dunin-Horkawicz, Stanisław; et al. (2013), "The RNase H-like superfamily: new members, comparative structural analysis and evolutionary classification", Nucleic Acids Research, 42 (7): 4160–4179, doi:10.1093/nar/gkt1414, PMC 3985635, PMrID 24464998

[12] 'Damerau–Levenshtein distance', 2017. [Online]. Available: https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance#Optimal_string_alignment_distance [Accessed: 09th June 2017]

[13] "Distance Measures for Sequences" by Sandeep Hosangadi