Ransomware Detection Considering User's Document Editing

Ransomware Detection Considering User's Document Editing

Toshiki Honda
Graduate School of Integrated Science and Technology
Shizuoka University
Shizuoka, Japan

Kohei Mukaiyama
Graduate School of Integrated Science and Technology
Shizuoka University
Shizuoka, Japan

Takeharu Shirai
Graduate School of Integrated Science and Technology
Shizuoka University
Shizuoka, Japan

Tetsushi Ohki
Graduate School of Science and Technology
Shizuoka University
Shizuoka, Japan
ohki@inf.shizuoka.ac.jp

Masakatsu Nishigaki
Graduate School of Science and Technology
Shizuoka University
Shizuoka, Japan
nisigaki@inf.shizuoka.ac.jp

***Abstract*— The number of victims suffering from crypto ransomware is increasing. Thus far, methods for detecting ransomware when it accesses target files or when it uses encrypting APIs, have been studied. However, the former method assumes it will be operated within an analysis sandbox, and the latter method can be avoided if the ransomware uses its own encrypting functions. To protect users, a detection method should be able to detect ransomware in the user's real-time environment and is difficult for the ransomware to avoid detection. This paper proposes a detection method that satisfies those requirements by using human file-operating characteristics as a whitelist. We evaluate the effectiveness of our prototype method, which inspects the consistency between the displayed documents and the user's editing operations.**

Keywords—ransomware, Encryptor, file protection, document editing, detection

I. INTRODUCTION

Ransomware attacks are occurring all over the world. The amount of crypto ransomware has grown in recent years. Crypto ransomware encrypts the victim's files and then demands a ransom for decrypting them. In several cases, victims have paid the ransom [1][2]. It is urgent to take measures to detect crypto ransomware. In previous studies, methods for detecting ransomware when it accesses target files or when it uses encrypting APIs, have been proposed. However, the former method is assumed to operate within an analysis sandbox and thus the latter method can be avoided if the ransomware uses its encrypting functions. To protect users, it is necessary to meet the following two requirements: (i) The ransomware must be detected in the user's real-time environment, and (ii) it should be difficult for the ransomware to avoid detection.

To solve these problems, we focus on the document-editing differences between humans and ransomware. Ransomware tries to encrypt files so as not to be noticed by the victim, regardless of the victim's actions. Therefore, its behavior reveals characteristics different from humans. In this paper, as the first step to our goal, we propose a new detection method that inspects the consistency between the displayed document contents and the user's document editing operations. By using human file-operating characteristics as a whitelist, the proposed method achieves to detect ransomware in the user's real-time environment and be difficult for the ransomware to avoid detection.

II. RANSOMWARE

Recent ransomware is generally classified into two types according to its attack method [3][4]: Locking and Crypto.

A. Locking Ransomware (a.k.a. Locker)

Locker locks a victim's computer, for instance, filling up the entire screen of the terminal with ransomware. Then demands a ransom to unlock it. In most cases, the victim can unlock it using utilities that disables the launching of Locker [5]. Booting the infected device into safe mode can also be used as a symptomatic treatment because the launching of third-party software is disabled in safe mode [6].

B. Crypto Ransomware (a.k.a. Encryptor)

Encryptor encrypts the victim's file or hard drive and demands a ransom for decrypting. An example of an attack is as follows.

1. Encryptor infects the victim's PC through a drive-by download attack or an email attachment.
2. It encrypts files in the background without the victim's awareness.
3. After the encryption is completed, the ransomware changes the wallpaper to an image that showing a ransom note and demands a ransom.

A decryption key is required to decrypt this type of encrypted files. Security vendors and the No More Ransom Project [7] analyze ransomware and, in some cases, publish a

decryption utility. If the utility is not published, however, it is impossible to decrypt the encrypted files without payment.

In addition, some ransomware removes the Volume Snapshot Service (VSS) files, which store periodic backup (a.k.a. shadow copies), so that victims cannot restore their computers [8]. The results of an Encryptor are more difficult to restore than a that of a Locker; thus, its threat is growing. In this paper, we focus on studying a method for detecting Encryptor.

III. Previous Works

A. UNVEIL

UNVEIL is a detection method suggested by Kharraz et al. that detects both Locker and Encryptor [9]. It utilizes file access patterns to detect ransomware. First, it generates an analysis environment that includes bait files that are ransomware targets. In this environment, file-system activities can be monitored by API hooking. Next, the malware targeted for analysis is launched in the environment. If UNVEIL finds following three situation, then the malware is detected as ransomware: multiple I/O requests related to writing or deleting the bait files; a significant increase in the entropy between read and write data buffers; or the creation of new high entropy files.

It is worth noting that this method cannot detect before files are encrypted. When this method detects an ransomware some files are already encrypted and/or locked. That is why, UNVEIL needs to assume to be operated in an analysis sandbox. Moreover, in this method, the ransomware can avoid detection by encrypting only a specific part of the file or destroying the file structure by swapping parts of the file.

B. Monitoring API Calls Relating to Encryption

Shigeta et al. found that Locky and CryptoWall, major Encryptors, use Microsoft CryptoAPI or OpenSSL. They suggested monitoring API calls that relate to encryption to detect the attacks[10]. In this method, Encryptors are detected when they have attempted to start file encryption. Files are protected by stopping the API execution by the operating system (OS) as soon as detection occurs. That is why, this method can be operated in the user's real-time environment. In this method, however, Encryptors can avoid detection by using other encryption libraries or implementing original encryption codes.

IV. Detection Method

A. Concept

In this section, we explain our proposed method. As mentioned in Section II.B, our method targets Encryptor because of its restoration difficulty. In addition, as described in Section III, previous methods have two problems: (i) UNVEIL is not be able to be operated in the user's real-time environment since it cannot detect Encryptors before files are encrypted, and (ii) the effectiveness of both methods may be limited since there are possible ways for Encryptors to avoid detection. In this paper, we aim for a solution to these problems and propose a detection method that uses user's document-editing characteristics as a whitelist.

By focusing on user's document-editing characteristics, we can detect Encryptors when a non-human Encryptor tries to manipulate a document. If detection occurs while an Encryptor is editing a document, then immediately the subsequent file access to reflect the contents to be edited on the document file is banned. Therefore, this type of detection method can protect the document file in the user's real-time environment.

The problem of ransomware avoiding detection is improved by adopting a whitelist-based detection method. Previous studies used a blacklist detection method that targeted ransomware features. In that method, the ransomware could avoid detection by finding a single loophole. In contrast, whitelist detection methods detect all attacks not included in the whitelist. Hence, it is more difficult for ransomware to avoid detection.

B. "Humanness" and "Ransomwareness"

When a user edits a document, the window of the software used to edit the document file is shown on the monitor. For example, when a user edits a text file, the Notepad program and the contents of the text file are shown. Fig. 1 shows an example of a user editing a document using Notepad. The user cannot edit a document without displaying the window and the file content. In this paper, we define this characteristic as "humanness."

In contrast, Encryptors must run in the background to encrypt a large number of document files without being noticed by the user. In other words, Encryptors do not show the window nor the document file content on the monitor. In this paper, we define this characteristic as "ransomwareness."

The critical characteristic is whether the software shows the content of the document files being edited. We propose a real-time Encryptor detection method that can proactively protect document files using these characteristics.

Fig. 1. Example of a display when a user is editing

C. Detection Method

1) Humanness Detection

In this paper, "humanness in document editing" is specified by the following two user's characteristics. When the detection method detects any file operation (see Section IV.A.1), it checks these characteristics (a) and (b). They are used as

whitelist to determine whether the file operation is being carried out by a user. To be explicit, as long as both of (a) and (b) are observed, the detector determines the current operator as a human.

(a) The monitor displays the document-editing software in an easy-to-see window size.

(b) The monitor displays document-editing software that contains the content of the document file.

As to (a), we define the minimum window size of Windows 7's Aero Snap feature as an easy-to-see size [11]. When a user drags a window to the corner of the screen, the window is resized to a quarter of the screen, as shown in Fig. 2. The size of the window in Fig. 2. is Aero Snap's minimum window size.

Fig. 2. Example of Aero Snap when a user drags a window to the monitor's upper right corner

As to (b), to check whether the window shows the document content, we classify the document as WYSIWYG or Non-WYSIWYG.

WYSIWYG (What You See Is What You Get). WYSIWYG document files contain font and layout information and the word-processing software parses and displays them. A Microsoft Word file is a typical example. Our method checks whether the file content is displayed in monitor by matching a screenshot image of the word-processing software window (hereafter, display image) and a simulated image reconstructed from the document file (hereafter, reconstructed image).

Non-WYSIWYG. Non-WYSIWYG document files do not contain their font and layout information. Therefore, the file content displayed by the word-processing software depends on its own settings, e.g., word wrap and fonts. A plain text file is a typical example. In our proposal, to acquire the text displayed on the window, we first take a screenshot image (display image), and then extract the text by processing the display image with optical character recognition (OCR). Our method checks whether the file content is displayed in monitor by matching the text extracted from display image and the text contained in the document file.

2) Detection Timing

Encryptor uses the following three I/O access patterns shown in Fig. 3 [9]. Each access pattern is as follows:

(1) Encryptor reads target file x and overwrites it with an encrypted version.

(2) Encryptor reads target file x and creates an encrypted version x.locked. Then, it deletes file x.

(3) Encryptor reads target file x and creates an encrypted version x.locked. Then, it overwrites file x it with an empty or garbled data to erase the content of file x.

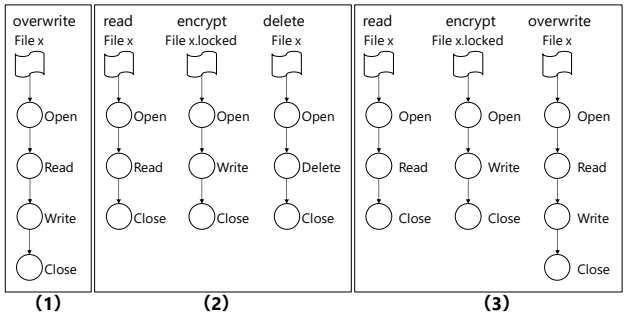

Fig. 3. Encryptor's I/O access patterns [9]

All of these patterns destroy the target file by deleting or overwriting file x. Therefore, an encryption by Encryptor can be prevented beforehand by prohibiting delete or overwrite operations for the file x if either of the behaviors (a) or (b) described in Section IV.C.1 is not observed. By doing so, it is expected that our method can achieve real-time detection in a user environment.

There are ransomware that also renames files before and after encryption. Therefore, in addition to the file x, it is necessary to monitor delete or overwrite operations for the renamed file x.renamed. Furthermore, the ransomware can generate an external process, such as PowerShell, to handle a part of its processing. Therefore, in the proposed method, it is essential to monitor all processes generated by the ransomware.

V. Implementation of the Proposed Method

We implemented a prototype of Encryptor detection program. As described in Section IV.C.1), documents are classified into WYSIWYG and Non-WYSIWYG, and there are a wide variety of file types in each of them. However, in order to simplify the implementation, the current detection program is only compatible with the detection of Encryptors targeting text files (*.txt). In addition, as described in Section IV.C.2, the proposed method needs to monitor all processes generated by the ransomware. However, also in order to simplify the implementation, the current detection program is only compatible with the detection of Encryptors operating in a single process.

A. Judge Whether the File Content Is Displayed

To determine whether the software is displaying the text content of file x, the following steps are required.

1. Obtain the character strings displayed on the monitor.
2. Extract the corresponding part of the file x.
3. Calculate the similarity between step 1 and step 2.

1) Obtain Character Strings Displayed on Monitor

To obtain character strings from the software window, the display image must be converted into character strings. Therefore, we employ OCR (Optical Character Recognition) [12] and extract the character strings from the display image. We need to understand here that OCR sometimes fails to convert them partly. Misrecognition will occur also because the software displays more than the file content (e.g., ruler, menu bar).

2) Extract Corresponding Part of File x

For a longer document file, the software shows the file content page by page, or part by part, on the monitor. Therefore, we must determine which character string in the document file x is displayed on the monitor before inspecting the consistency between the displayed document contents and the user's document editing operations. We employ the LCS (Longest Common Subsequence) [13] from the Diff utility to look for the corresponding part in the file contents. Fig. 4(i) shows an example of the result of character string extraction using LCS.

3) Calculate Similarity

We employ the Jaro-Winkler distance [14] to calculate the similarity between the character string obtained by OCR (Step 1) and the displayed file content (Step 2). The Jaro-Winkler distance represents the distance between two strings from 0 (least similar) to 1 (most similar). Fig. 4(ii) shows an example of the result of the Jaro-Winkler distance calculation.

We conducted a preliminary experiment to determine the similarity threshold. In the preliminary experiment, we observed the Jaro-Winkler distance value in both cases that the correct file content is displayed and incorrect file content is displayed using randomly chosen 10 text files. The maximum value was about 0.78 when the correct file content was displayed, and the maximum value was about 0.66 when incorrect file content was displayed. Therefore, we set the threshold as 0.7.

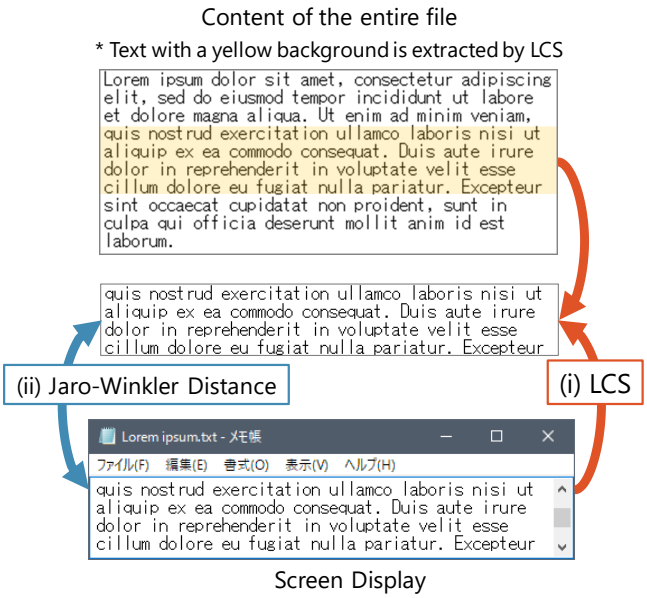

Fig. 4. Example of checking whether the file content is displayed

B. Detection Program

The detection program is a resident program that monitors all API calls made by each software, and judges whether or not the software is an Encryptor. More specifically, with the timing when a software executes a delete/overwrite operation for an existing file or the renamed file (Section IV.C.2), it checks whether the software's behavior deviates from the humanness characteristic (Section IV.C.1) using the Jaro-Winkler distance between displayed content and file content (Section V.A). When a deviating software is detected, the detection program identifies it as Encryptor and prevents it from deleting or overwriting the files, and immediately displays an alert.

To monitor API calls from the target software, the detection program injects a DLL (dynamic-link library) file into each software process [15]. The detection method is composed of a DLL injector, which injects a DLL into the software, and a DLL file, which monitors API calls and detects Encryptors (Fig. 5).

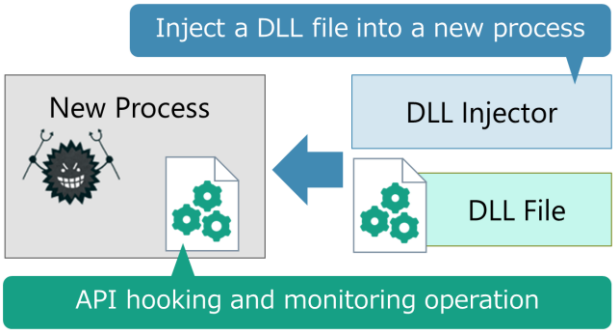

Fig. 5. Outline of the detection program

1) DLL Injection

The DLL injector monitors new processes every 500 ms, and injects a DLL file into them.

2) Operations of DLL File

The DLL file operates as described below.

(1) The detection program monitors API calls made by the process for reading any existing text file. (Please remember that our current detection program tries to detect Encryptors targeting text files.) When a text file is renamed by the process, the detection program considers the renamed file as a text file and includes the renamed file in the target files to be monitored.

(2) When the process reads a text file, the detection program reads the same text file and obtains the character strings of the file content.

(3) The detection program takes a screenshot of the software window and converts it into character strings using OCR. In addition, the detection program judges whether the monitor is displaying the document-editing software in an easy-to-see window size as described in Section IV.C.

(4) The detection program compares the character strings obtained in (2) with the character strings extracted in (3), using the method shown in Section V.A, and

calculates the Jaro-Winkler distance to see whether the text file is displayed on the screen.

(5) If the detection program judges that the process is not Encryptor, the process can delete or overwrite the file read in (1). If the detection programs judges the process as an Encryptor, deleting and overwriting are prohibited.

Ideally, these operations (1)-(5) should be executed when a process requests a file deletion or overwrite. However, it turned out that processing (2)-(4) took an amount of time. Therefore, in our detection program, the operations are started when any process generates a file-read API call. Thus, these operations are conducted before a file deletion/overwriting occurs, and if once Encryptor is detected, any subsequent executed file deletion and overwriting made by the process (Encryptor) are forbidden.

VI. Evaluation

We performed detection experiments and false-detection experiments and evaluated the effectiveness of the proposed method.

A. Detection Experiment

1) Purpose

We evaluate the effectiveness of the proposed detection method by checking whether Encryptors can encrypt files while the detection program is running.

2) Experimental Procedure

We observed what happens to text files on an infected PC when the detection program was running and when it was not running. We used Hybrid Analysis [16] to collect ransomware samples, namely, Cerber, Jaff, WannaCry, and Locky.

This experiment was conducted on QEMU, which is a virtual PC emulator. Table I shows the experimental environment. Using a blind text generator [17], we created 40 text files with various combinations and numbers of characters. Fig. 6 shows 1000.txt, an example of the created text files. We placed 20 text files and one folder on the desktop and the remaining 20 text files inside the folder. A folder named "Prog," containing the detection program, was also located on the desktop. To prevent the detection program from being encrypted during ransomware operation, the "Prog" folder was set to "read only." All ransomware samples are located in the Download folder.

TABLE I. Detection Experiment Environment

QEMU	2.5.0 Debian 1:2.5+dfsg-5ubuntu10.7
Guest OS	Windows 8.1
Resolution	1,024×768
Memory	2,048 MB
Internet Connection	None (Network Adaptor is connected)
OS Language	English
Windows Update	Until 2017/2/3 17:00
Windows Defender	Disabled

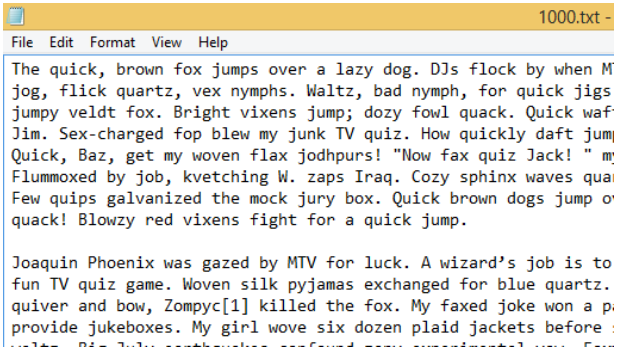

Fig. 6. Part of the file contents of 1000.txt

3) Results

Table II shows the results when text is infected with each Encryptor. The detection program detected Cerber and Jaff, and it was able to prevent those file encryptions. However, WannaCry and Locky could not be detected or prevented. Due to page limit, here provide more details about Cerber and WannaCry.

TABLE II. Results for Each Family of Detection Experiments

Family	Generates other processes	Detection	Avoid Encryption	Avoid Rename after Encryption
Cerber		✓	✓	
Jaff		✓	✓	
WannaCry	✓			
Locky	✓			

During our experiment, in case the PC was infected with Cerber while the detection program was not running, the file names of two text files, including 1000.txt, were changed. In addition, the wallpaper was changed to an image demanding a ransom (Fig. 7). We observed the contents of the two renamed files with Notepad. The beginning part of the file was the same as before running Cerber, but the rest had been encrypted. Fig. 8 shows JrofdGp16O.a49e, which was a file renamed 1000.txt. Comparing Fig. 6 with Fig. 8, it can be observed that the content of 1000.txt was encrypted.

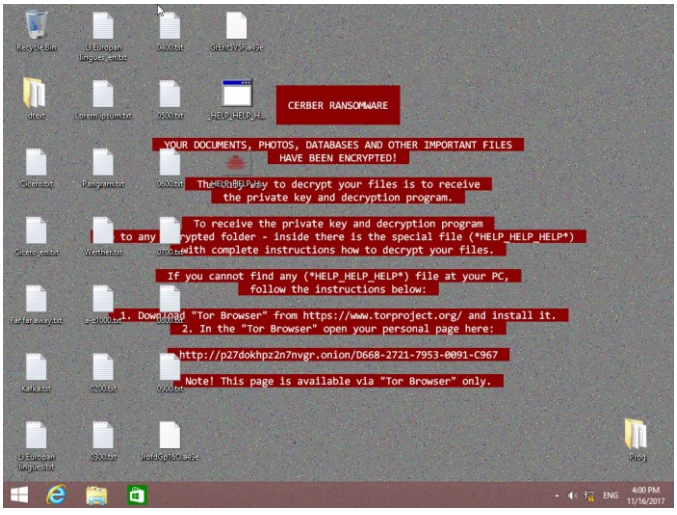

Fig. 7. Desktop after ransomware infection

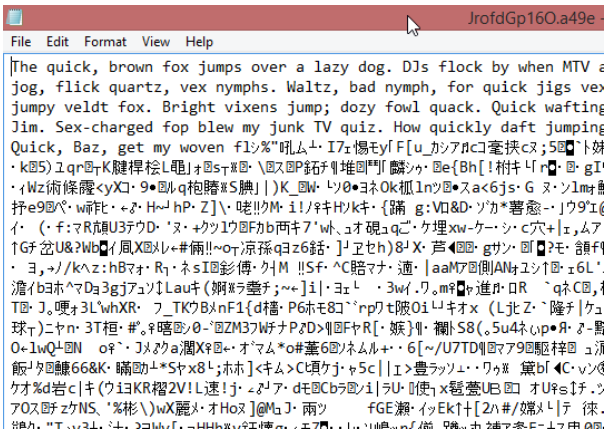

Fig. 8. Part of the file contents of JrofdGp16O.a49e

When the PC was infected with Cerber while the detection program was running, a message announcing that ransomware was detected was displayed. Cerber was able to rename two text files, including 1000.txt, and change the wallpaper the same as mentioned earlier. However, we observed that the two renamed files had the same file contents as before the infection. Fig. 9 shows R2AlxP3MQu.a49e, which was a file renamed 1000.txt. Comparing Fig. 6 with Fig. 9, we observed that the content of 1000.txt was not encrypted, and the file contents were successfully protected.

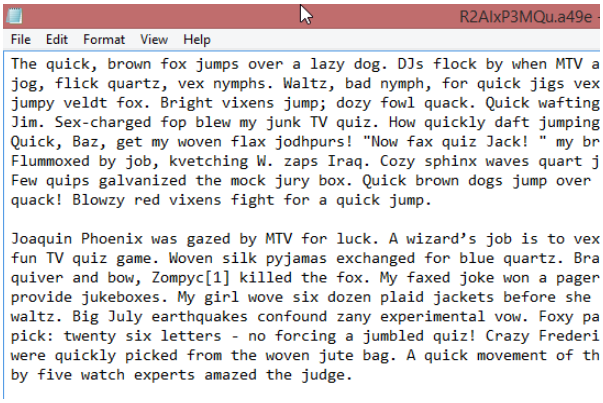

Fig. 9. Part of the file contents of R2AlxP3MQu.a49e

To clarify the reason why our detection program failed detection of WannaCry and Locky, we investigated how WannaCry encrypts files and found that WannaCry do not operate in a single process. WannaCry invokes a system process and get help with (a part of) crime from the co-process. It is impossible to hook system processes by DLL injection and therefore the current detection program cannot monitor the operation of multiple processes as mentioned before.

This was same with Locky. Therefore, we can say that our evaluation results confirmed the effectiveness of the proposed method against Encryptor that runs in a single process.

B. *False-Positive Experiment*

1) *Purpose*

When the user edits a text file while the detection program is running, we confirm that the user operations are not erroneously detected as ransomware operations.

2) *Experimental Procedure*

We prepared text files with various strings of two to 10,000 characters and opened them in Notepad with the detection program activated. We used the "Pangram" dummy text of the blind text generator[17] to generate the contents of the file. Notepad's window size was set to one quarter of the screen size (the minimum size shown in Section IV.C.1). After that, the user added the letter "A" to the beginning of the text and overwrote it after approximately three seconds. Table III shows the experimental environment.

TABLE III. FALSE-POSITIVE EXPERIMENTAL ENVIRONMENT

Machine	OS	Windows 10
	Screen Resolution	1,920×1,080
Notepad	Version	1607
	Word Wrap	Enabled
	Font	Consolas
	Font Size	11pt
	Window size	929×540

3) *Results*

Table IV summarizes the results of the false-positive experiment. "✓" indicates that the overwrite was permitted and "X" indicates that the overwrite was rejected; i.e., a false-positive detection as ransomware. It should be noted that in all cases, overwriting was allowed. It was confirmed that the Jaro-Winkler distance always exceeded the threshold value of 0.7 when a user is editing a file. Based on the results, it is expected that false-positive detection of editing by users will not occur.

TABLE IV. OVERWRITING JUDGMENT FOR EACH NUMBER OF CHARACTERS

Number of Characters	Jaro-Winkler Distance	Overwrite
2	0.822222	✓
4	0.933333	✓
10	0.950000	✓
50	0.848188	✓
100	0.878358	✓
500	0.805686	✓
1000	0.886816	✓
10000	0.890940	✓

VII. DISCUSSION

A. *Detection Avoidance by Ransomware*

The proposed method recognizes ransomware by whether the file contents are being displayed on the screen. Therefore, to avoid the proposed method, the ransomware must display the target-file contents on the screen for a certain period when encrypting it. This raises the risk of the ransomware being noticed by the user.

The ransomware might avoid detection by displaying the file content in a small window that is not easily noticeable by the user. In the proposed method, the minimum window size is defined in Section IV.C.1. By refusing to allow the software to overwrite the file if the displayed document-editing window is

less than the minimum window size, it is expected that the ransomware will find it difficult to encrypt without being noticed by the user.

Alternatively, the ransomware could prepare a dummy file, display it on the screen, and encrypt the target file in the background. However, when another document file is displayed on the screen, the similarity between the character strings of the file contents and the character strings displayed on the screen becomes lower (Fig. 10). Thus, detection avoidance by displaying dummy contents does not work effectively.

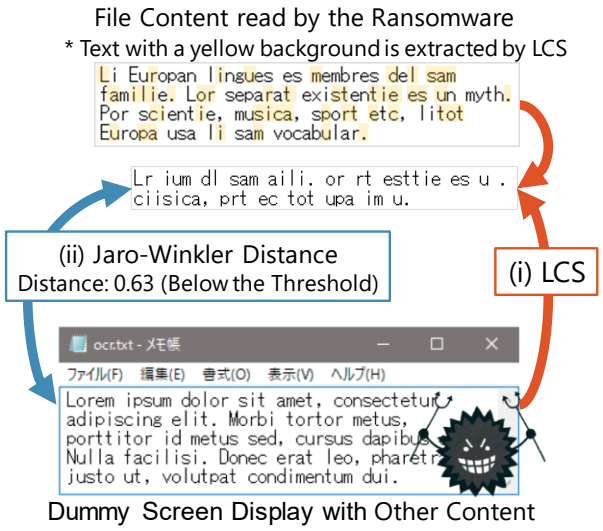

Fig. 10. Sample distance calculation when a dummy file is displayed

B. *Implementation of Detection Program*

In this paper, DLL injection was adopted to implement the detection program. However, certain ransomware operates in multiple processes. As it is impossible to perform API hooking to system processes, the detection program does not work effectively in such software. To address this problem, the detection method should be implemented as an OS function and monitor the I/O accesses and screen displays of all processes.

C. *File Operation Exception*

The proposed method is a whitelist-type ransomware detection method that uses the user's document editing as a feature representing "humanness." Therefore, a false-positive detection occurs for software that performs a file operation without displaying the contents of the file, e.g., general encryption software or file converter software. In such software, although the file contents are not displayed, information, e.g., the file path and file name, usually will be displayed.

In addition, the user will provide input to the software, e.g., select a file or click a button. By including these in the definition of "humanness," we believe it is possible to reduce the false-positive detections for software that manipulates files without displaying the file content.

Moreover, we believe that the proposed method is applicable not only for text document but also for a variety type of contents (e.g., Image Editor, Excel, PowerPoint, etc.), by using sophisticated image matching instead of OCR.

A critical weakness of the proposed method is that there are legitimate programs that do not display file contents, such as batch processes, and the proposed method will make misjudge. This is the future work we must address.

VIII. Conclusion

In this study, we focused on the difference between "humanness" and "ransomwareness" and proposed a whitelist-type ransomware-detection method. The proposed method prevented files from being encrypted by crypto ransomware by restricting the file deletion and overwriting. As a first step, we implemented a ransomware-detection program to protect text files and evaluated its effectiveness.

We confirmed that when the detection program operates as suggested in our concept, it can detect ransomware and prevent the encryption of the document files. In addition, false-positive detections do not occur as shown in our experiment of editing by a user using Notepad.

Our future tasks are as follow:

- Confirm the effectiveness of the proposed method for other ransomwares.
- Implement a detection program targeting WYSIWYG document files.
- Address problem mentioned in Section VII.C.

Acknowledgment

The authors wish to acknowledge Associate Professor Atsushi Nakazawa of the Graduate School of Informatics, Kyoto University for advice in selecting the algorithm for image matching the WYSIWYG document file.

References

[1] Richard Winton: Hollywood hospital pays $17,000 in bitcoin to hackers; FBI investigating, Los Angeles Times, available from 〈 http://www.latimes.com/business/technology/la-me-ln-hollywood-hospital-bitcoin-20160217-story.html〉 (accessed 2017-12-03).

[2] University of Calgary paid $20K in ransomware attack, CBC News, available from 〈 http://www.cbc.ca/news/canada/calgary/university-calgary-ransomware-cyberattack-1.3620979〉 (accessed 2017-12-03).

[3] Kevin Savage, Peter Coogan, Hon Lau: The evolution of ransomware, available from 〈 http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-evolution-of-ransomware.pdf 〉 (accessed 2017-12-03).

[4] Marvin the Robot: Infographic: What you need to know about ransomware, Kaspersky Lab official blog, available from 〈https://www.kaspersky.com/blog/ransomware-infographics/13315/〉 (accessed 2017-12-03).

[5] Anastasiya Angel: Tip of the week: Fighting screen lockers, Kaspersky Lab official blog, available from 〈 https://www.kaspersky.com/blog/kaspersky-windowsunlocker-2/12275/〉 (accessed 2017-12-03).

[6] John Zorabedian: Android "police warning" ransomware - how to avoid it, and what to do if you get caught, naked security, available from

〈 https://nakedsecurity.sophos.com/2014/05/19/android-police-warning-ransomware-how-to-avoid-it-and-what-to-do-if-you-get-caught/〉 (accessed 2017-12-03).

[7] The No More Ransom Project, available from 〈https://www.nomoreransom.org/〉 (accessed 2017-12-03).

[8] Paul Ducklin: Got ransomware? What are your options?, naked security, available from 〈 https://nakedsecurity.sophos.com/2016/03/03/got-ransomware-what-are-your-options/〉 (accessed 2017-12-03).

[9] Kharraz, A., Arshad, S., Mulliner, C., Robertson, W. and Kirda, E., "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," 25th USENIX Security Symposium, 2016, pp.757-772.

[10] Takanari Shigeta, Masakatu Morii, Tomohisa Hasegawa, Masato Ikegami, Teiichi Ishikawa, "Encryption Processing of Ransomware," Computer Security Symposium 2016, pp.134-137 (in Japanese).

[11] Miron Vranjes: Arrange your Windows in a Snap, Windows Experience Blog, available from 〈 https://blogs.windows.com/windowsexperience/2015/06/04/arrange-your-windows-in-a-snap/〉 (accessed 2017-12-03).

[12] zdeno: Tesseract Open Source OCR Engine, available from 〈https://github.com/tesseract-ocr/tesseract〉 (accessed 2017-12-03).

[13] Wu, S., Manber, U., Myers, G., and Miller W., "An O(NP) Sequence Comparison Algorithm," Information Processing Letters, Vol.35, No.6, 1990, pp.317-323.

[14] Winkler, W. E., "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage," Proceedings of the Section on Survey Research Methods. American Statistical Association, 1990, pp.354–359

[15] Brad Antoniewicz: Windows DLL Injection Basics , Open Security Research , available from 〈 http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html〉 (accessed 2017-12-03)

[16] Hybrid Analysis, available from 〈https://www.hybrid-analysis.com/〉 (accessed 2017-10-16).

[17] BLIND TEXT GENERATOR, available from 〈http://www.blindtextgenerator.com/〉 (accessed 2017-04-17).