Methods to Hide Malicious Codes in PowerPoint

Jiwoong Choi, Inhwan Kim, Seunghee Han, Sungmin Lee, and Jooseok Song

***Abstract*—During several years, the number of the malicious code and the methods for hiding them highly increases. The disseminating the malicious code commonly abuses general files in a way of hiding to avoid its existence from being detected. Among general files, the document files (DOC, PPT, XLS, PDF, etc.) can be easily used for malicious purposes. Therefore, we analyze the PowerPoint 2003 file based on PowerPoint 2003 file format document because it is commonly used by host users. In this paper, we briefly describe the PowerPoint 2003 file format and propose the methods that can be used to store the malicious code. In experiments, we verify whether the proposed methods work well or not.**

***Index Terms*—Malicious code, virus, PowerPoint.**

I. Introduction

The Malicious code is one of the executable codes that aim to damage target host for malicious purposes. As IT technology is developed, malicious code has been made for many purposes such as disclosing personal information, occurring network traffic that are not intended and deleting system files, etc. [1]. Furthermore, the number of the malicious code increases [2], [3] and the methods for disseminating the malicious code became more complicated and sophisticated to avoid its existence from being detected by anti-virus detection. One of the most efficient methods to disseminate the malicious code is adding the code into a general file which has the space to include it. Common user cannot realize its existence because hiding in a general file makes the malicious code undetectable or the existence secret. To avert suspicion, attackers usually use the file such as DOC, PPT, XLS, PDF, etc. [2]-[4] which can easily pretend normal and unsuspicious. Among these files, MS-OFFICE document file is usually used. For example, a macro virus is a program written in the macro language included in Word document files [4]-[6]. When a Word document file that is infected by a macro virus is opened, the macro virus will copy itself into other file or make a new file which can be executed and infect the target system. It means other files subsequently could be infected if the infected document file is opened once. In MS-OFFICE document files, there are several formats (Word, Excel, PowerPoint, etc.). Among them, we have found the methods to make the space in PowerPoint 2003 file which can include malicious code.

TABLE I: Stream and Storage of PowerPoint 2003

Name	Contents
Current User stream	The most recent UserEdit offset of PowerPoint Document stream
PowerPoint Document stream	Most of the document data
Pictures stream	Embedded picture data
Summary Information stream	Meta data for the document
Document Summary Information stream	Meta data for the document
Encrypted Summary Information stream	Encrypted Summary Information stream
Digital Signature storage	Embedded signature data
Custom XML Data storage	Embedded XML data
Signatures stream	Embedded signature data

According to our results, the space can have data with variable length and PowerPoint 2003 application does not recognize its existence. Since PowerPoint 2003 file can be used for malicious purpose, we recommend that the space must be checked in anti-virus detection.

The rest of this paper is organized as follows. PowerPoint 2003 file format is discussed in Section II. The methods to store the malicious code in PowerPoint 2003 file are presented in Section III. Then the methods are experimented in Section IV.

II. PowerPoint 2003 Structure

In this section, we show PowerPoint 2003 file format which is described in the PowerPoint 2003 file format document [7].

A. Properties

PowerPoint 2003 file is following the OLE (Object Linking and Embedding) compound file format [8], [9]. In the OLE compound file format, the document file is composed of "stream" and "storage" which are binary data in a hierarchical structure. Like file and directory of the file system, stream and storage take the similar role respectively. Storage can contain either stream or storage in a hierarchical structure. Stream and storage contain a document file data and the names of them are different depending on each document file format. The stream and storage that compose a PowerPoint 2003 file is as follows in the Table I. Among them, the PowerPoint Document stream contains most of the document file data and it consists of a lot of "record". And records are classified into two types, "atom" and "container" which is similar to stream and storage. Like storage, container can contain either atom or container. In other words, the PowerPoint Document stream is composed of

Manuscript received June 14, 2013; revised August 2, 2013. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012 R1A1B3004161).

Jiwoong Choi, Inhwan Kim, Seunghee Han, Sungmin Lee, and Jooseok Song are with the Computer Science Department, Yonsei University, Seoul, Republic of Korea (e-mail: {hope0371, ihkim, seungheeh, niji, jssong @emerald.yonsei.ac.kr).

a collection of records in a hierarchical structure and each record refers a piece of the document file.

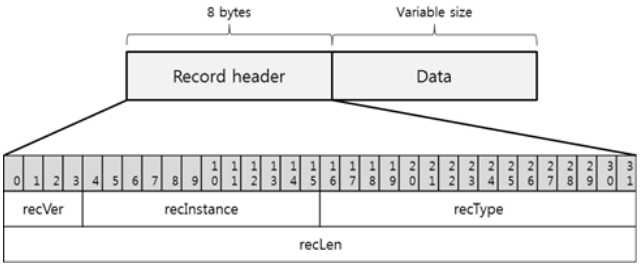

Fig. 1. Record structure.

B. Record Structure

PowerPoint 2003 file format document describes many records that contain various meaning of a document file data. Fig. 1 shows the record structure [7]. Record structure is divided into record header and data. In front of a data part, a record header is placed. The values in a record header are used to identify and interpret a record data that follows. Thus, each record has a record header which interprets what a record contains in a record data.

C. UserEdit Structure

The PowerPoint Document stream is composed of one or multiple UserEdits which is conceptual unit and represents changes made by user. The PowerPoint Document stream has multiple UserEdits if the function "Allow fast saves" is enabled in PowerPoint 2003 application [10]. In this case, UserEdit is created newly or modified from the previous UserEdit and it is appended to the PowerPoint Document stream whenever the document file is written. Furthermore, UserEdit is composed of a set of particular records that are called "persist object" containing primary information of the document file. The persist object is a top-level object that can be independently persisted and includes a unit of a document file data such as slide, main master slide, notes, handout, etc.

For example, SlideContainer record represents the slide data and NotesContainer represents the notes data. Fig. 2 shows the interpretation flow of the PowerPoint Document stream for UserEdit unit.As described in Table I, the data of Current User stream represents where the most recent UserEdit is. And each UserEdit stores the offset for previous UserEdit except for the top of UserEdit whose start offset is 0 when UserEdit exists more than one. So, the interpretation of the PowerPoint Document stream starts from Current User stream. After finding the offset of the most recent UserEdit, each UserEdit is found by the UserEdit chain sequentially. The Interpretation flow of the PowerPoint Document stream that includes multiple UserEdits for record unit is performed as Fig. 3 [7], [10]. Each UserEdit has one UserEditAtom and one PersistDirectoryAtom. UserEditAtom contains the metadata about UserEdit such as the offset for previous UserEdit, the offset for PersistDirectoryAtom and the file version, etc. PersistDirectoryAtom contains a persist object ID and offset pairs. A persist object ID is assigned to a persist object to distinguish itself from other persist objects. Therefore, each persist object in UserEdit can be found via PersistDirectoryAtom.

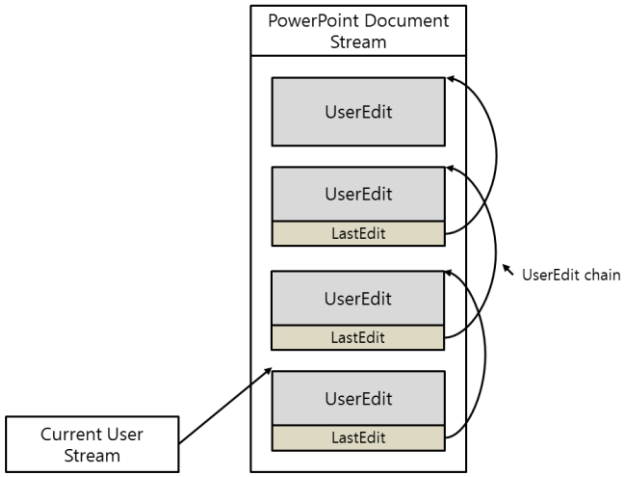

Fig. 2. Interpretation flow of the PowerPoint document stream for UserEdit unit.

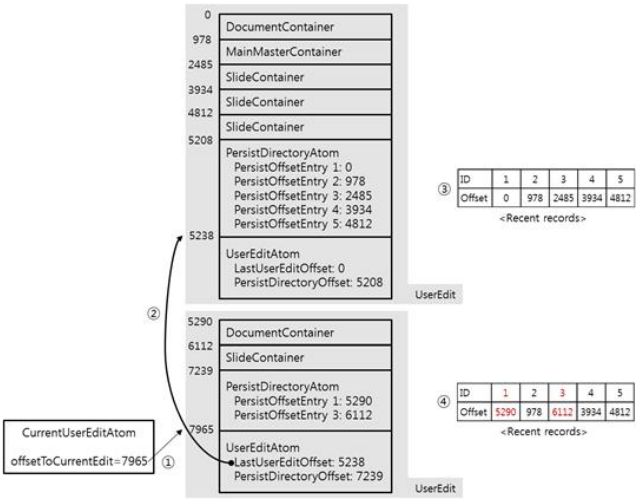

Fig. 3. Interpretation flow of the PowerPoint document stream for record unit.

1) The start point that is the offset of the most recent UserEditAtom is gained from CurrentUserEditAtom of CurrentUser stream. Then read the most recent UserEditAtom.
2) If the offsetLastEdit field of the identified UserEditAtom is not 0, then read the previous UserEditAtom sequentially using the offsetLastEdit field until the offsetLastEdit field of the identified UserEditAtom is 0.
3) When the UserEditAtom whose offsetLastEdit is 0 is found, the corresponding PersistDirectoryAtom is read through the offsetPersistDirectory field of the identified UserEditAtom. The persist object ID and offset pairs are stored to the memory that we call recent records. Then, the pairs from each PersistDirectoryAtom that were identified in previous steps are read in reverse order and continually added to the recent records.
4) During adding a new pair to the recent records, if the persist object ID already exists in the recent records, the persist object offset from the new pair replaces the existing persist object offset for that persist object ID. When the all PersistDirectoryAtom are checked, the pairs in the recent records represent current data of the document file. Finally, using the pairs in the recent records, recent persist objects are interpreted

III. Methods to Store Malicious Codes

We developed the methods to make the spaces to store malicious codes in PowerPoint 2003 file. The methods that we have found are using previous UserEdit and the UnknownBinaryTag record.

A. The Method of Using Previous UserEdit

As we mentioned above, the PowerPoint Document stream may have multiple UserEdits and it seems not to have any other problem. However, there are some persist objects (records) that have not examined what it contains in interpretation process. We should notice that it can be possible for the previous UserEdit to be used to add some data unrelated with the document file. In interpretation process of the PowerPoint Document stream that has multiple UserEdits, when the duplicate persist object ID is discovered, it just deletes the old offset and adds the new one. This action deserves more than a passing notice because it just identify the data at the new offset without examining what it contains at the old offset. In other words, malicious codes can be inserted in the persist object at the old offset (the duplicate offset) and PowerPoint 2003 application does not examine the data. So, attacker can insert malicious codes into previous UserEdits as following steps:

1) Enable the function "Allow fast saves" in PowerPoint 2003 application and then make new file [10].
2) Modify the file more than once.
3) Find the persist objects in previous UserEdits among multiple UserEdits and replace the data of the persist objects with malicious codes.

B. The Method of Using UnknownBinaryTag Record

There is a name/value pair structure called programmable tags or ProgTags that contains document file data. The programmable tags are included in several container records as Table II [7]. DocProgTagsContainer contains the setting values for document level such as prohibited words, fonts, default text, etc. SlideProgTagsContainer contains the setting values for slide level such as slide creation time, linked slide, comments, etc. ShapeProgTagsContainer contains the setting values for shape level such as character-level formatting, paragraph-level formatting, etc. Since the supported functions are different in each PowerPoint application version, the programmable tags have been made to apply the setting values depending on the PowerPoint application version. According to the name of programmable tags, the setting values could be either applied or ignored on each PowerPoint application version. Therefore, PowerPoint 2003 file format document presents the container records that contain programmable tags and the names of programmable tags that should be permitted on each PowerPoint application version as Table III [7]. As described in Table III, the container records may contain not only the names that have to be applied but also UnknownBinaryTag record. If the PowerPoint application encounters a name of a programmable tag that is not in the permit name list, it interprets the programmable tag as UnknownBinaryTag record. Otherwise, the programmable tag refers to KnownBinaryTag record. As the PowerPoint 2003 file format document describes, UnknownBinaryTag record MUST be ignored and MUST be preserved regardless of what it contains in the data of programmable tags. This means malicious codes can be inserted to UnknownBinaryTag. So, the method of using UnknownBinaryTag record can be achieved as following steps:

1) Add new programmable tag or modify the existed programmable tags with the name that is not in the permit list. Then modify the length field of the record header for name part to appropriate value leaving the other fields unchanged.
2) Replace the data of the value part with malicious codes. Then modify the length field of the record header for value part to appropriate value leaving the other fields unchanged.
3) Modify the offset value for other records where PeristDirectoryAtoms follow UnknownBinaryTag.

TABLE II: Kind of Programmable Tags

Record	Description
DocProgTagsContainer	Settings for document level
SlideProgTagsContainer	Settings for slide level
ShapeProgTagsContainer	Settings for shape level

TABLE III: Name List of Programmable Tags

Record	Name	Description
DocProgTags Container	___PPT9	PPT 97 ignores it
	___PPT10	PPT 97, 2000 ignore it
	___PPT11	PPT 97, 2000, 2002 ignore it
	___PPT12	PPT 97, 2000, 2002, 2003 ignore it
	Any other value	UnknownBinaryTag
SlideProgTags Container	___PPT9	PPT 97 ignores it
	___PPT10	PPT 97, 2000 ignore it
	___PPT12	PPT 97, 2000, 2002, 2003 ignore it
	Any other value	UnknownBinaryTag
ShapeProgTags Container	___PPT9	PPT 97 ignores it
	___PPT10	PPT 97, 2000 ignore it
	___PPT11	PPT 97, 2000, 2002 ignore it
	Any other value	UnknownBinaryTag

IV. Experiments

The methods have been experimented using POI (Poor Obfuscation Implementation) and OffVis (Office Visualization Tool) [11], [12]. POI is the JAVA libraries that help in analyzing the MS-OFFICE document file formats. OffVis provides functions that can modify and parse OLE document hierarchically.

Firstly, the method of using previous UserEdit has been experimented by modifying the SlideContainer in previous UserEdit whose persist object ID is duplicate. Fig. 4 shows both original data and modified data. As we described before, the duplicated persist object does not identified and examined in the interpretation process. So, even though we modified the whole data of the SlideContainer, the document file is opened well without any notice of modification on the PowerPoint 2003 application.

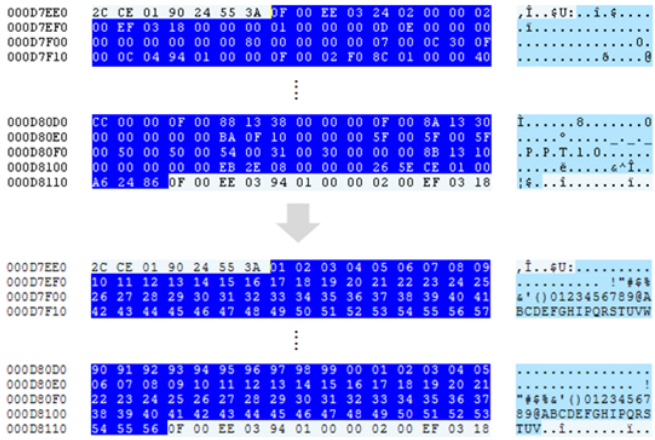

Fig. 4. Modification of previous UserEdit.

Secondly, the experiment for the method of using UnknownBinaryTag record has been done by modifying the existing DocProgTagsContainer. We modified the programmable tags whose name is "___PPT10" excluding the record header data. In Fig. 5, bottom one shows modified data with random binary values. As a result, we can say that the UnknownBinaryTag record is created. Even though we added the UnknownBinaryTag record with random binary values that have no relevance with the document file, the document file is opened without any problems on the PowerPoint 2003 application.

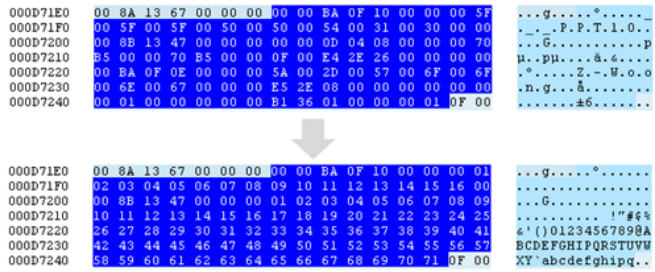

Fig. 5. Modification of KnownBinaryTag to UnknownBinaryTag.

V. Conclusion and Future Works

In conclusion, we presented two possible methods to insert malicious codes in PowerPoint 2003 file by analyzing the PowerPoint 2003 file format document. According to our result of experiments, although the binary data that have no relation with the document file are included, the document file works well with no problems. So, we recommend that the space that we discovered must be checked on anti-virus tool or the malicious code detection.

In future works, there are other MS-Office document file formats such as Word, Excel and these files are also very commonly used. Therefore, we will analyze other document file formats and find the method to make the space that can be used to add malicious codes. Moreover, we will make the methods to detect the space that we find.

References

[1] A. Sprykin, V. Kiktenko, S. Galagan, and A. Kunitsky, "Malicious code testing in executable files," in *Proc. Modern Problems of Radio Engineering, Telecommunications and Computer Science Conf.*, Lviv-Slavsko, February 2008, pp. 632-634.

[2] K. Selvaraj and N. F. Gutierrez, "The rise of PDF malware," *Symantec Corporation*, June 2010.

[3] Y. X. Gao and D. Y. Qi, "Analyze and detect malicious code for compound document binary storage format," in *Proc. 2011 Machine Learning and Cybernetics Conf*, Guilin, July, 2011.

[4] V. Bontchev, "Possible macro virus attacks and how to prevent them," *Computers & Security*, vol. 15, no.7, pp. 595-626, 1996.

[5] W. J. Li, S. J. Stolfo, A. Starvrou, E Androulaki, and A. D. Keromytis, "A study of malcode-bearing documents," in *Proc. Detection of Intrusions and Malware & Vulnerablility Assessment (DIMVA) Conf.*, July 2007, pp.231-250.

[6] J. Munro. (July 2002). Antivirus research and detection techniques. *Extreme Tech*.. [Online]. Available: http://www.extremetech.com/article2/0%2C1558%2C325439%2C00.asp

[7] PowerPoint (.ppt) Binary File Format, MicroSoft Co., February 11, 2013.

[8] Compound File Binary File Format, MicroSoft Co., January 18, 2013.

[9] Apache Open Office. Microsoft compound document file format. [Online]. Available: http://www.openoffice.org/sc/compdocfileformat.pdf

[10] J. Park and S. Lee, "Forensic investigation of Microsoft PowerPoint files," *Digital Investigation*, vol. 6, pp.16-24, September 2009.

[11] Apache Software Foundation. Apache POI open source project. [Online]. Available: http://poi.apache.org

[12] Microsoft Co. Microsoft office visualization tool (OffVis). [Online]. Available: http://www.microsoft.com/en-us/news/press/2009/jul09/07-27BlackHat09PR.aspx

Jiwoong Choi received the B.S degree in Computer Science from Kyonggi University, Suwon, Republic of Korea, in 2011. He is currently working toward the M.S. degree in Computer Science at Yonsei University, Seoul, Republic of Korea. His research interests include information security and wireless ad hoc networks.

Inhwan Kim received the B.S and M.S degree in Computer Science from Ajou University, Suwon, Republic of Korea, in 2007 and 2009, He is currently working toward the Ph.D. degree in Computer Science at Yonsei University, Seoul, Republic of Korea. His research interests include location security in vehicular ad hoc networks and denial of services in wireless networks.

Seunghee Han received the B.S degree in Computer Science from Yonsei University, Wonju, Republic of Korea, in 2011, He is currently working toward the Ph.D. degree in Computer Science at Yonsei University, Seoul, Republic of Korea. His research interests include enhancement throughput and wireless networks security.

Sungmin Lee received the B.S degree in Computer Science from Inha University, Incheon, Republic of Korea, in 2007, He is currently working toward the M.S. degree in Computer Science at Yonsei University, Seoul, Republic of Korea. His research interests include network security and cryptography.

Jooseok Song received the B.S. degree in Electrical Engineering from Seoul National University, Republic of Korea, in 1976, and the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology, Republic of Korea, in 1979. In 1988, he received the Ph.D. degree in Computer Science from University of California at Berkeley.

From 1988 to 1989, he was an assistant professor at the Naval Postgraduate School, Monterey, CA. He was the president of Korea Institute of Information Security and Cryptology in 2006. He is currently a Professor of computer science at Yonsei University, Seoul, Republic of Korea. His research interests include cryptography and network security.