applied sciences

MDPI

Article

Malware Detection on Byte Streams of Hangul Word Processor Files

Young-Seob Jeong, Jiyoung Woo and Ah Reum Kang *

SCH Media Labs, Soonchunhyang University, Asan 31538, Korea; bytecell@sch.ac.kr (Y.-S.J.); jywoo@sch.ac.kr (J.W.)
* Correspondence: armk@arkang.net

Received: 30 October 2019; Accepted: 27 November 2019; Published: 29 November 2019

Abstract: While the exchange of data files or programs on the Internet grows exponentially, most users are vulnerable to infected files, especially to malicious non-executables. Due to the circumstances between South and North Korea, many malicious actions have recently been found in Hangul Word Processor (HWP) non-executable files because the HWP is widely used in schools, military facilities, and government institutions of South Korea. The HWP file usually has one or more byte streams that are often used for the malicious actions. Based on an assumption that infected byte streams have particular patterns, we design a convolutional neural network (CNN) to grasp such patterns. We conduct experiments on our prepared 534 HWP files, and demonstrate that the proposed CNN achieves the best performance compared to other machine learning models. As new malicious attacks keep emerging, we will keep collecting such HWP files and investigate better model structures.

Keywords: malware detection; HWP file; byte stream; convolutional neural network

1. Introduction

As Internet users exchange more data files or programs, the threat of malicious code is also on the rise. The malicious code may exist in executable files or non-executables (e.g., PDF files, Word documents). The Internet users are usually aware of the danger of apocryphal executables, but many of them are not conscious of the non-executables. If a user opens a contaminated document, then a malicious code of the document will download executable files that may cause serious damage. For instance, it may steal important information (e.g., bank account) of the user or request ransom money for the data stored on the computer. Although existing security programs help to detect such malicious actions, it is hard to put the malicious code out of existence because new malicious attacks keep emerging. The new appearing attacks are mostly capable of detouring the existing security programs, so it is important to watch and keep developing improved detection methods.

Hangul Word Processor (HWP) is a word processing application provided by Hancom Inc., South Korea. The HWP is one of the most widely used pieces of software in South Korea, and its main customers includes schools, military agencies, and governmental institutions. The HWP document files are non-executables, and the customers are vulnerable to contaminated HWP files. Because of the political and military situation between South and North Korea, many contaminated HWP files have been found and their main target was the governmental institutions of South Korea. The malicious HWP files usually contain byte streams of executable codes, shell codes, and script codes. The byte streams of malicious actions are much different from that of benign HWP files, so it is possible to detect the malicious actions by analyzing the byte streams of HWP documents.

The detection of malicious action is basically a binary classification problem; it determines whether a given file contains any malicious actions or not. There are various machine learning models (e.g., logistic regression, support vector machine [1], random forest [2]) known to be effective for the classification problem. Although these models have shown successful performance (e.g., accuracy) on various classification tasks, they have a common limitation that they require an intensive effort to define high-quality features. If the features are poorly defined, the overall performance will be severely degraded. Deep neural network is one of solutions to address such a limitation, as it automatically extracts effective features. The neural networks of recent studies can be categorized into two types: recurrent neural network (RNN) [3] and convolutional neural network (CNN) [4]. The RNN has a recursive connection in a temporal manner that allows it to effectively understand sequential patterns, while the CNN finds arbitrary local patterns and generates a high-level representation by summarizing the found patterns. The CNN is also known to be efficient as it has a relatively smaller number of parameters to learn.

In this paper, we aim at detecting malicious actions of the HWP files. We do not put much effort into defining features, but feed the byte sequences of HWP files as an input to our designed CNN model. This is based on the assumption that there must be latent patterns in the byte sequences of malicious actions. The CNN model is designed to find such patterns and summarize them, so that it finally determines whether the given sequence has any malicious actions or not. To the best of our knowledge, this is the first study to detect the malicious actions of HWP files by analyzing the byte sequences. We believe that this study will help to prevent the customers (e.g., governmental institutions of South Korea) from damage driven by malicious HWP files.

2. Background

2.1. Malware Detection on Stream Data

Malware is a program designed to harm or disturb computer systems. It usually steals important information of the computer user or request ransom money for maliciously encrypted data. Malware can be divided into two categories: executables and non-executables. The danger of malicious executables (e.g., exe files) is widely known by the public, and there are many security programs (e.g., Norton, Kaspersky) to detect malicious actions of the executables. On the other hand, the non-executables (e.g., docx, HWP files) often bypass some existing security programs and there is a high risk of false positives. Moreover, the non-executables tend to be considered as being insignificant by many users, which makes them more dangerous.

There have been studies to detect the malicious actions of non-executables. Srndic and Laskov [5] proposed a format-free hierarchical document structure, namely Hidost, to represent structural paths of non-executable documents, and train machine learning models with features about the obtained paths. Cuan et al. [6] defined features using PDFiD Python script that verifies objects in PDF files. Because the features might be vulnerable to a simple trick such as a gradient-descent attack, they also introduced two feature selection steps: (1) applying a threshold for each feature, and (2) reducing the number of non-critical features. Smutz and Stavrou [7] defined features obtainable from document metadata, such as the length of title and image sizes. Li et al. [8] proposed a feature extractor for PDF documents, namely FEPDF, that generates features about a document header, objects, cross-reference, and trailer. All of these studies mainly put much effort into finding effective features based on their knowledge about the target file. It is expensive to design high-quality features, and different feature definitions will be required for different types of documents.

In this paper, we aim at detecting malicious actions within Hangul Word Processor (HWP) file that is a widely used non-executable format in South Korea. Due to the political and military situation between South and North Korea, one of main targets of malicious HWP files is the governmental institutions of South Korea. As far as we know, this is the first study that detects malicious actions of the HWP files by analyzing byte sequences. We design a convolutional neural network that determines whether a given byte sequence contains malicious actions or not. This is based on an assumption that the byte sequences of malicious actions contain particular patterns. Our proposed network is intended to automatically extracts effective features about the patterns. Although the proposed network does not require intensive feature definition, it is still important to examine and understand the structure of the target format (i.e., HWP). The reason is that the better features can be generated only if input data is wisely defined by domain experts, and better understanding of the file structure might help to design better networks.

2.2. Stream Data of HWP Files

Malicious non-executable documents take advantage of features and vulnerabilities provided by an office software (e.g., Word, HWP). In order to detect malicious HWP Files, it is essential to understand the HWP document format and figure out how the malicious code is inserted into HWP structure. HWP document is organized into a hierarchy of storage and streams. It consists of FileHeader, DocInfo, BodyText, Summary Information, BinData, PrvText, PrvImage, DocOptions, Scripts, XML Template, and DocHistory, as shown in Table 1. The FileHeader contains a HWP signature. The DocInfo contains details commonly used within a document, such as fonts, text attributes, and paragraph attributes. The BodyText contains texts such as paragraphs, tables, and drawing objects. The document summary information includes information such as the HWP document title, author, creation date, and last modified date. The BinData stores images, Object Linking and Embedding (OLE) objects, and post scripts as separate streams in BinData storage. The PrvText stores the preview text as a Unicode string, while the PrvImage is stored in the bitmap image (BMP) or Graphic Interchange Format (GIF). The DocOptions store linked documents, distribution documents, Digital Rights Management (DRM), and digital signature information in separate streams. The Scripts have JavaScript codes stored as streams in Scripts storage. The Extensible Markup Language (XML) Template contains template information, and DocHistory has document history management information.

There are various strategies to attack: object injection, JavaScript, Visual Basic for Application (VBA) macro, HWP vulnerabilities, and Encapsulated PostScript (EPS). For the HWP documents, the object injection attack inserts abnormal values or Portable Executable (PE) codes into tags representing text of paragraphs in BodyText storage, or malicious streams may be included in BinData storage. The JavaScript attack inserts malicious JavaScript codes into the Script section. The VBA can be used to describe macro that is an executable command contained within a document. The HWP vulnerability exploits a software flaw in Hangul Word Processor program. As an example, an overflow vulnerability in HwpApp::CHncSDS_Manager function used in HWP could affect the execution flow of a program through large paragraph documents. The attackers mainly take the EPS-based attack, where the EPS format represents graphical images using the PostScript programming language created by Adobe. Since the EPS can express various high-definition vector images, the HWP provides a function to include or view EPS images in HWP documents. The attackers make these EPS files malicious and include them in HWP documents.

In Figure 1, a partial example of an EPS stream obtained from a malicious HWP document is shown. This is BIN0008.eps stream in BinData storage, which is compressed with deflate compression algorithm. The size of the stream is 25 KB before compression and 5 KB after compression. This stream can be decompressed using deflate compression algorithm, and the result is depicted in Figure 2. Figure 3 is an example of a normal EPS file, and obviously it is different from Figure 2. The code in Figure 2 has a variable with a long string called *ar*, and there is a code to operate XOR and execute it. The variable *ar* obfuscates a shellcode and includes the malicious behavior for downloading files from external sites. Normal EPS code does not need obfuscation to hide a shellcode, too long variables, XOR operations, or command *exec*, as shown in Figure 3.

Table 1. Hangul Word Processor (HWP) document structure.

Description	Storage	Stream
File recognition information		FileHeader
Document information		DocInfo
Main text	BodyText	Section 0 Section 1 ...
Document summary		I HWPSummaryInformation
Binary data	BinData	BinaryData0 BinaryData1 ...
Preview text		PrvText
Preview image		PrvImage
Document option	DocOptions	_LinkDoc DrmLicense ...
Script	Scripts	DefaultJScript JScriptVersion ...
XML template	XMLTemplate	Schema Instance ...
Document history management	DocHistory	VersionLog0 VersionLog1 ...

```
äWKrÃ ½JoÐÄ5 1Ó Ò=z…Þ Q ÁÂ!N ™f2þÄ<é = †÷ïŸ·/îðŠ ¯ 8J b Ñëˆ>3œè÷Ì
‰ÞûŠ ë G ;ìH¸†ò ƒ7P\¬ŒÑ< ½×Q˜Í€ 8Ýâ×U ý†Ù U¹Ã02! ÐÀ VmDôÙf#kDMxU
‰>smÓ„~éôî·¹5°!ÇÒE/ SÝAy}©Ì`_D°Å€Ît@1yõÒã â ¹á9NµSTB² “
ÚØLï£ W}W$yt{¶ È!/È¨6e )õE,*tÁ—ž,Ûô>Fî«
“™ —qÙ–} …"ù2ÍKY5kjìD5vÙepí)\Ö›p R¨¾ ¥ a Á‘uŠgÆßÀMµíSl“MÊÛ~î¥’¶?1ü
%}N{dÃ£ UóØT1hTŒ¤èoUÌ+WU‘«uW]c™{ø®Ê+üj®Ì |ˆO ù*5ú # 2Vjøˆ±¡²eÞíŠ¯
þîªM6 VIv¾ · 6© '×™×Vì‰×™&ÕÉT O(Š  (>O 6eÏ î™WÎð =Ó¤B¿;×˜gˆü<lž Æs
```

Figure 1. Example of an Encapsulated PostScript (EPS) stream in a malicious HWP file.

```
/ar <E2A1ACAFA8A1FCEDFCFBEE8B8B8B8BEDA9A8ABEDE2A1ACAFA8A1
FFEDA1ACAFA8A1FCEDACBFBFACB4EDA9A8ABEDE2A1ACAFA8A1FEED
E5BDA2A2BFE4EDA9A8ABEDE2A1ACAFA8A1F9EDFCEDACBFBFACB4ED
A9A8ABEDE2A1ACAFA8A1F8EDFDE … 1A8EDBCB8A4B9C7C0C7>
def /limit {ar length -1 add}
def /len {ar length}
def /str len string
def ar 0 1 limit {
        2 copy get 205 xor put ar
}for
pop str 0 1 limit {
        dup ar exch get put str
}for cvx exec exec
```

Figure 2. Example of a decompressed EPS stream in a malicious HWP file.

```
/ms {moveto show} bind def
/s /show load def
/SF {
        findfont exch scalefont dup setfont def
} bind def
/sf /setfont load def
/rect {
        4 2 roll moveto
        1 index 0 rlineto
        0 exch rlineto
        neg 0 rlineto
        closepath
} bind def
/BeginEPSF {
        /b4_Inc_state save def
        /dict_count countdictstack def
        /op_count count 1 sub def
        userdict begin
        /showpage { } def
        0 setgray 0 setlinecap
        1 setlinewidth 0 setlinejoin
        10 setmiterlimit [ ] 0 setdash newpath
        /languagelevel where
                {pop languagelevel
                1 ne
                {false setstrokeadjust false setoverprint
                } if
        } if
}bind def
/EndEPSF {
        count op_count sub {pop} repeat
        countdictstack dict_count sub {end} repeat
        b4_Inc_state restore
} bind def
/pgsave save def
/F1 40 /Times-Roman SF
...Set some text with F1...
/F2 40 /Times-Italic SF
...Set some text with F2...
F1 sf
...Set some more text with F1...
F2 sf
...Set some more text with F2...
BeginEPSF
65.2 10 translate
.80 .80 scale
-4 -4 translate
EPS
4 4 604 403 rect
clip newpath
```

Figure 3. Example of an EPS code in a normal HWP file.

We also examined malicious macro streams and VBA macro codes. Figure 4 shows an example of a macro stream of a malicious HWP file. This stream is BIN0002.OLE stream in BinData storage and is compressed with deflate compression algorithm. The size of this stream is 37 KB before compression and 12 KB after compression. This stream exploits the macro functionality provided by HWP. This stream can be decompressed using deflate compression algorithm, and the result is depicted in Figure 5. Figure 6 is a VBA macro code called Laroux in the late 90ś and extracted from the malicious HWP document. When an infected file is opened, the Laroux creates PERSONAL.XLS file in XLSTART folder of Microsoft Excel. This malicious code does not show any characteristic symptoms, but there are various errors among the variants, and some errors occur over time.

Through the examination to the malicious HWP documents, we observed that the malicious byte streams have different appearance from the benign streams. That is, the malicious codes have to employ particular functions (e.g., eval, exec, xor) to trigger malicious actions, and the byte streams reflect such behaviors. This implies that it is reasonable to take the byte streams as features. We also observed that the streams of malicious actions mostly do not exceed 600 bytes, and actual length of malicious actions are only 10 $\sim$ 200 bytes. Small fractions of malicious actions may be scattered throughout a stream, which makes difficult to detect the malicious actions. Thus, it is essentially necessary to collect more HWP files conveying distributed malicious codes and investigating them.

```
ìW pTçu>ÿÝ• „ ‹´z ¤ÕÕJ X=0â) Iø ÞÂ Â ,ap$$ Q›Ä kãL;nÚq,'™Î¸3NÇ™I'6Å ¿
6,` 6.Ø<Lc»&i »µkãÄ6nŠ~ç¿ï»{m4“f¦™Þ ³÷ÞóŸ÷9ÿùÏ¥ï =3éò ž(ü%¹®Eä£ÏÇR)Ù†
xf W€HÁ áó±±1 Çÿ¿þ Eà3ØZ‰ä- Ì $ 8ç .)€T@ `" È di%@“.Ù€ @ È ä.“ …€)€"@
1 ( ¨€R@ P ( T.¦ ¦ ¸®nÀ}:` `&  P ˜ ˆ.j u€z@ƒÎ3 ÷6@#`.` `>` `!`   Ð h , ´ Úuþ õ;ëÿS¿
VÓ.~ÃÈE;mÇ}ˆvÁûë¿òP1FŒ¸ ”¦ð?Ñˆ&,ËÊ¼ ~íá N c8ñ¸  >ê½~….Ê4R„=G×+(¨
d~€úi vtÓ ×ËnÒeC?÷D?€í¸ L¿Y'ì„ö!ú:ts ¾~=ì Ö?^ÿÙÖ|=3\ãF gü—í ¦ÿcíÿJØS Ïþç
Æ=ƒ{@#à‹ö?ûbßÿm ç À¡á p àfÀ-€¯.– – – V.V V V :.k.· Ö : ë.ë _ I.D _ t 6 ¸ º · z.
```

Figure 4. Example of a macro stream in a malicious HWP file.

```
le“`P. .MSFormsC P. VBAProject¾¿P. Office uP. Module1b P. €..ÿ ._Evaluate ÙP. larouxW¾P.
auto_openV P. .Application¥*P. .OnSheetActivateúnP. check_files $P. .cZ P. .StartupPathiôP. .md
P. .pg P. .ActiveWorkbook ¢P. .Modules„ P. .Count0vP. .wm P. .whichfileëcP. .ScreenUpdating!
P. .n4O^P. .SheetsP. .Visible¶ÓP. .CopyÆ¿P. .Titleò~P. .SubjectRPP. .Authorü P. .Keywords
TP. .CommentsÃ)P. .newname#P. .c4¸\P. .ActiveWindowÃ+P. .WorkbooksBìP. .SaveAsf;P. .Filena
mejÃP..FileFormatŸÖP. .xlNormalLP. .PasswordðÎP. .WriteResPassword :P. .ReadOnlyRecommen
dedíŸP. .CreateBackup P. .p4™^P.
```

Figure 5. Example of a decompressed macro stream in a malicious HWP file.

```
Sub auto_open()
  Application.OnSheetActivate = "check_files"
End Sub
Sub check_files()
  c$ = Application.StartupPath
  m$ = Dir(c$ & "\" & "PERSONAL.XLS")
  If m$ = "PERSONAL.XLS" Then p = 1 Else p = 0
  If ActiveWorkbook.Modules.Count > 0 Then w = 1 Else w = 0
  whichfile = p + w * 10
Select Case whichfile
  Case 10
  …
  If s$ <> "laroux" Then
    Workbooks("PERSONAL.XLS").Sheets("laroux").Copy before:=Workbooks(n4$).Sheets(1)
    Workbooks(n4$).Sheets("laroux").Visible = False
  Else
  End If
  Application.OnSheetActivate = ""
  Application.ScreenUpdating = True
  Application.OnSheetActivate = "personal.xls!check_files"
  Case Else
End Select
End Sub
```

Figure 6. Example of a Visual Basic for Application (VBA) macro code in a malicious HWP file.

2.3. Neural Networks for Malware Detection

Deep neural network is drawing attention mainly for two reasons: (1) it automatically extracts features without much effort of domain experts, and (2) it has shown its superior performance (i.e., accuracy) over other machine learning models in many tasks such as object recognition [9], text classification [10], and machine translation [11]. There have been few studies that apply the neural networks to malware detection. The studies are basically based on two distinct analysis ways: dynamic analysis and static analysis. The studies of dynamic analysis are conducted under the binary run in a virtual environment (e.g., virtual box). For example, Kolosnjaji et al. [12] proposed to combine long short-term memory (LSTM) [13] with convolutional filters in order to classify malware types, where the features were obtained from API call sequences through the virtual environment. Huang and Stokes [14] also utilized the API calls and original function calls to define 114 features. They proposed two distinct models for malware detection and malware type classification, respectively, and insisted that sharing parameters of the two models improves the overall performance. In [15], the recurrent neural network (RNN) is employed to capture sequential patterns of abnormal behavior of executables. Xiao et al. [16] proposed a deep learning framework of stacked auto-encoders (SAE) [17] that utilizes behavioral patterns to detect malware. These studies of dynamic analysis commonly require a virtual environment to simulate the executables, and it is difficult to reproduce their works because different analysis is usually conducted on a different non-public emulation environment.

There were studies of static analysis that define features obtainable without running the malicious codes. Raff et al. [18] defined features of 300 raw-bytes of the PE-header of each file, and feed them to the neural networks to detect malicious actions. This work showed that the neural networks have a potential to interpret the raw-bytes and generate high-level representation without much effort of feature engineering. Saxe and Berlin [19] defined a fixed-length feature vector based on a histogram of byte entropy values, and feed them to train the neural networks. Le et al. [20] proposed CNN-BiLSTM architecture, where the CNN captures local patterns and the bi-directional LSTM (BiLSTM) layer finds sequential dependencies between the found patterns. Raff et al. [21] derived a feature vector from the entire byte sequence of each file, and designed a shallow convolutional neural network. The shallow network takes input sequence of extraordinary length (e.g., 1–2 M length), and insisted that the network will be available to variable length of byte sequence; however, it is still not be applicable to any longer sequences (e.g., 3–4 M lengths). All of these studies of static analysis commonly aimed at detecting malicious actions of executables.

There have been only a few studies of static analysis to detect malicious actions of non-executables. Jeong et al. [22] recently designed a shallow convolutional neural network (CNN) to analyze the byte sequences of PDF files. They assumed that there must be scattered patterns representing malicious actions of byte sequences, and chose to employ the CNN because the CNN is known to be effective in capturing local promising patterns. Inspired by the study, we design a CNN to capture arbitrary patterns of malicious actions by analyzing the byte sequences of HWP files. The biggest difference between [22] and this study is the different characteristics of target files (HWP files versus PDF files). It is important to understand the structure of target files even if we adopt deep learning models because better understanding of the target files will help to design a better structure for solving the problem. For example, the proposed model in this paper has two FC layers whereas the model in [22] has one FC layer. This can be interpreted that the HWP files might contain more complicated relationships among promising local values.Indeed, the malicious actions of HWP files take various types (e.g., JavaScript, Visual Basic for Application (VBA), and Encapsulated PostScript (EPS)), whereas most of the malicious actions in PDF files use Java script. A malicious HWP file often contains multiple types of attack (e.g., VBA with object injection, HWP vulnerability with PE codes), and the second FC layer of our model may contribute to extract relational information between the different types of attack.

3. Proposed Method

Our problem is to detect malicious actions within a given byte sequence of a target HWP file, and we design a convolutional neural network (CNN) as a solution to tackle the problem. Although employing the CNN means that we do not have to put much effort for feature definition, it is still important to understand the structure of the HWP file. That is, domain knowledge is still required to wisely define input data (e.g., input dimension) and design networks for better performance (e.g., accuracy). As shown in Figure 7, a HWP file may have multiple byte streams, where a benign stream does not contain any malicious actions, and a malicious stream has one or more malicious actions. The malicious actions are often scattered to multiple positions. By manual examination of the scattered malicious actions, we found that at least one of them has a particular malicious action (e.g., download from arbitrary links) while other remains usually support it in various ways (e.g., camouflaging by irrelevant functions, jump to the offset of malicious action). We also found that the length of a malicious action usually does not exceed 600 bytes. Based on these findings, we design a CNN to capture and summarize such patterns, as depicted in Figure 8.

The input vector of the proposed CNN is a byte sequence of length $S = 600$. The input vector is firstly delivered to the embedding layer that projects each nominal value of the vector into a E-dimensional space. Formally, S dimensional input vector is projected into a $E \times S$ matrix, where each i-th column vector corresponds to i-th byte of the input sequence. The embedding layer is widely adopted in the area of natural language processing because it incorporates semantic pattern of the nominal values. For example, the words 'buddy' and 'friend' will be close to each other in the embedding space (e.g., E dimensional space), whereas 'friend' and 'enemy' will appear at totally different positions. Similar to this, the E-dimensional representation of our embedding layer will incorporate semantic patterns of the byte sequences. In this study, we found that the optimal E is 64.

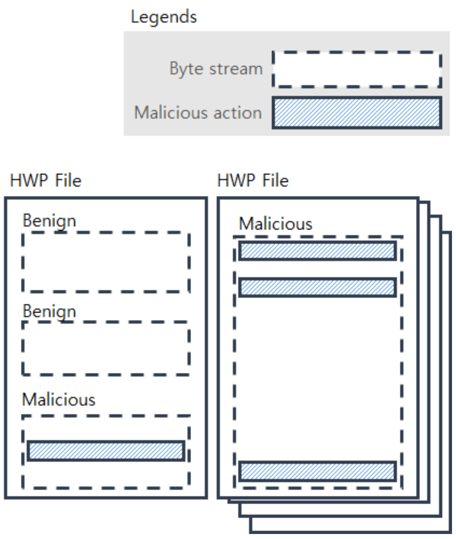

Figure 7. Malicious actions within byte streams of HWP files.

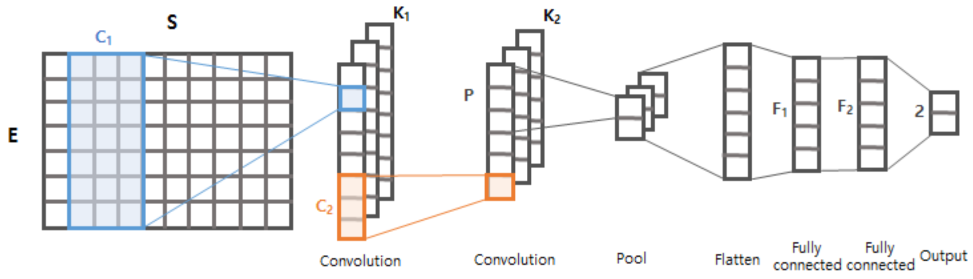

Figure 8. Graphical representation of the proposed convolutional neural network (CNN).

The $E \times S$ matrix generated by the embedding layer is passed to the first convolutional layer. The convolutional layer has K_1 convolutional filters, where the filter size is $C_1 \times E$. The convolutional filter slides from left to right of the $E \times S$ matrix, and used to mask each $E \times C_1$ matrix in order to find arbitrary local patterns. As each filter generates an output vector independently, different filters will probably capture different local patterns. The K_1 output vectors generated by the first convolutional layer is then fed to the second convolutional layer. The second convolutional layer works similar to the first one, where it has K_2 convolutional filters of size $C_2 \times 1$. The intuition behind the two consecutive convolutional layers is that the first layer captures local patterns of action functions (e.g., print, eval) and the second layer will generate high-level features by interpreting the relationship between the functions.

The second convolutional layer generates K_2 output vectors as high-level features, and they are passed to a pooling layer. Similarly to the convolutional layers, the pooling layer of size P slides through the given vectors, but it summarizes the vectors by picking the most promising (e.g., maximum) value amongst every P values. The outputs of the pooling layer are flattened to form a vector, which is delivered to two consecutive fully connected (FC) layers. These two layers are supposed to find relationships between the picked promising values, where they generate F_1 and F_2 dimensional vectors, respectively. Finally, the output layer takes the F_2 dimensional vector as an input, and generates 2 dimensional vectors for the final decision.

4. Experiment

As there is no available public dataset of HWP files, we prepared 534 HWP files (benign: 79, malicious: 455) for experiments. Unlike the benign HWP files, the malicious HWP files use various strategies to attack; so we collected more malicious data to incorporate such diversity. We believe that our dataset covers diverse ways of malicious attacks, and hope that the dataset will help many other related studies. We sampled a set of streams from the HWP files through the Algorithm 1, where $\boldsymbol{D}$ refers to the set of HWP files, K is the number of sampling for malicious case, and L is the number of sampling for benign case. The sampled streams have the same length S = 600. The statistic of the sampled data is described in Table 2. For every experiment in this study, we employ precision, recall, F1 score, and accuracy for measurement, and take 10-fold cross validation. We used a machine having Intel(R) Core(TM) i7-7700 CPU 3.60 GHz, two Geforce GTX 1080 Ti, and 16 GB RAM.

Algorithm 1: Sampling benign or malicious streams

```
Result: The set of sampled streams $\boldsymbol{R}$
$S \leftarrow 600$; $K \leftarrow 20$; $L \leftarrow 3$; $i \leftarrow 0$; $\boldsymbol{R} \leftarrow$ empty list [];
while $i \neq |\boldsymbol{D}|$ do
    $d \leftarrow \boldsymbol{D}_i$;
    if $d$ is malicious then
        $m_{index} \leftarrow$sequence index of malicious action;
        Sample $K$ randomly chosen indices $I_{malicious}$ between $[m_{index}, m_{index}+400]$;
        for $k \leftarrow$ each index of $I_{malicious}$ do
            $\boldsymbol{R}$.append($d$[k, k + $S$]);
        end
    else
        Sample $L$ randomly chosen indices $I_{benign}$ between $[0, |\boldsymbol{D}| - S]$;
        for $k \leftarrow$ each index of $I_{benign}$ do
            $\boldsymbol{R}$.append($d$[k, k + $S$]);
        end
    end
    $i \leftarrow i + 1$;
end
```

Table 2. Statistics of sampled data.

Total	Malicious	Benign
13,181	7031	6150

We define the raw values of the byte streams as features, meaning that each data instance is represented as an S dimensional feature vector. Because the values of the feature vector are nominal values, we generate a dictionary of the nominal values and transform the values of feature vectors into the dictionary indices. Assuming that we have a dictionary {0: 'FF', 1: 'FE', 2: 'BC', 3: '0A'}, a byte sequence ['FF', 'FE', '0A', 'FF'] will be transformed into a vector [0, 1, 3, 0]. Formally, the set of sampled byte streams $\boldsymbol{R}$ generated from the Algorithm 1 is transformed into the set of feature vectors $\boldsymbol{F}$ using the dictionary.

We compare our proposed CNN with several traditional machine learning models such as Gaussian naive bayes (GNB), logistic regression (LR), and random forest (RF). The GNB is a naive bayes classifier with Gaussian distribution, that allows to adjust to unseen values. The LR is a probabilistic classifier using linear combination of independent variables. The RF is a kind of ensemble model that generates final result by incorporating results of multiple decision trees. Through a grid search, we optimize the parameter settings of the models as described in Table 3.

Table 3. Description and parameter settings of comparative classifiers.

Model	Description
Gaussian naive bayes (GNB)	- Naive bayes classifier with Gaussian distribution
Logistic regression (LR)	- Probabilistic classifier using linear combination of variables - penalty = L2 - C = 2.0 - Solver = liblinear - Maximum iteration = 100
Random forest (RF)	- Ensemble model incorporating results of multiple decision-trees - # of estimators = 20 - Criterion = gini algorithm - Minimum # of samples to split = 2

For our CNN model, by grid searching, we found that the optimal structure is as follows: $E = 256$, $C_1 = 3$, $C_2 = 3$, $K_1 = 32$, $K_2 = 64$, $P = 100$, $F_1 = 32$, and $F_2 = 32$. We define a cost function as a cross entropy over the output layer, and the total number of trainable parameters is 70,178. The training recipe can be summarized as follows: (1) L2 gradient clipping with 0.4, (2) drop-out [23] with a keeping probability 0.5 for the fully connected layers, (3) Adam's optimizer [24] with an initial learning rate 0.001, and (4) a mini-batch size = 100. We also tried regularization methods (e.g., L2 and decov [25]), but no performance gain was observed; the reason might be that the drop-out plays a role as a regularization method. All the weight matrices and bias vectors are initialized by He's algorithm [26] and zeros, respectively.

Table 4 summarizes the performance comparison of different models. As our problem is essentially a binary classification, there are two values (for benign and malicious classes) for the F1 score, precision, and recall. For the models utilizing random processes (e.g., random forest, CNN), we take average values of several (e.g., three times) independently conducted experiments. For fair comparison, all models are trained using the same features of raw byte sequences. Among the three traditional machine learning models, the logistic regression (LR) achieved the best performance. Obviously, the proposed CNN exhibits better performance than all the other models; the accuracy gap between the CNN and the LR is almost 5%. Because it will be dangerous if we miss only a single malware, the higher recall of malicious cases (i.e., sensitivity) should be preferable. In this sense, the LR must be the most preferable model because of its overwhelmingly better recall (e.g., 98.57%). However, its precision of malicious cases is the worst among all models; it may cause some important files to be thrown out even though they are benign. As the recall of CNN for malicious cases is moderate and its accuracy is the best, the CNN might be chosen if we seek the most stable and accurate model.

Table 4. Performance comparison of different models, where the two values b/m of each cell correspond to benign and malicious cases, respectively.

Model	Accuracy	F1	Precision	Recall
GNB	81.31	81.79/80.80	74.98/89.35	89.95/73.74
RF	80.18	82.17/77.68	78.98/81.83	85.64/73.94
LR	88.43	88.00/88.83	98.45/80.84	79.56/98.57
CNN	**93.39**	**93.33 / 93.45**	**88.22 / 99.09**	**99.07 / 88.42**

5. Discussion

One might argue that there can be better structures than our proposed CNN. As described so far, we conducted a grid search to find the optimal structure, as well as the best training regime. Several samples of grid searching are shown in Table 5. We tried various structures of less or more layers, and found that the two consecutive convolutional layers followed by a pooling layer and two fully connected layers are the optimal. We also conduct, of course, various settings of the training recipe (e.g., batch normalization [27], drop-out [23], decov [25], and gradient clipping), but we do not put all cases into the table due to the limited space. When we take account of the F1 score of malicious cases, our optimal setting achieved 93.45%, as shown in Table 4, which is absolutely better than the other remaining settings, as shown in Table 5.

Table 5. Samples of different CNN structures, where the two values b/m of each cell correspond to benign and malicious cases, respectively.

Structure	**F1**
Embed($E = 512$) + Conv($K_1 = 32, K_2 = 64$) + Pool($P = 100$) + FC($F_1 = 32, F_2 = 32$)	92.61/92.63
Embed($E = 128$) + Conv($K_1 = 32, K_2 = 64$) + Pool($P = 100$) + FC($F_1 = 32, F_2 = 32$)	93.00/93.07
Embed($E = 256$) + Conv($K_1 = 32$) + Pool($P = 100$) + FC($F_1 = 32, F_2 = 32$)	92.52/92.50
Embed($E = 256$) + Conv($K_1 = 32, K_2 = 64$) + Pool($P = 100$) + FC($F_1 = 128$)	92.63/92.63
Embed($E = 256$) + Conv($K_1 = 64, K_2 = 64$) + Pool($P = 100$) + FC($F_1 = 32, F_2 = 32$)	92.97/92.99
Embed($E = 256$) + Conv($K_1 = 32, K_2 = 8, K_3 = 8$) + Pool($P = 100$) + FC($F_1 = 32, F_2 = 32$)	92.47/92.43

We believe that our study can prevent the public from malicious actions of HWP files. For example, a company mailing system may check every attached HWP file before the users download and open it. To do so, the proposed model should be fast enough to deal with a plenty of HWP files. We check the elapsed time for making predictions for 2061 instances, and compare it with other models as shown in Table 6. We observe that the LR is the fastest, whereas the slowest model is GNB. Although the total amount of elapsed time might not be linearly proportional to the number of instances, we can conclude that the proposed CNN is applicable to the scenario of company mailing system, as the runtime for a single instance is about 0.82 milliseconds.

Table 6. Runtime comparison, where the elapsed time is specified in seconds.

Model	**Runtime (Seconds)**
GNB	3.3028
RF	0.0976
LR	0.0026
CNN	1.6988

We performed an analysis of failed cases and identified two reasons. The first reason is a small number of samples. We collected only 455 malicious files: 286 files of EPS, 51 files of VBA macro, 3 file of JavaScript, 104 files of iframe tags, and 11 file of HWP vulnerabilities. The samples extracted from the HWP vulnerabilities or JavaScript methods are relatively small compared to others, so they become a weakness of the trained model. The second reason is the fixed length of streams. In many of the samples we collected, we manually identified the presence of malware in 600-byte segments and chose it as the stream length. However, in some cases, short code fractions are scattered throughout the corresponding steam. To address this limitation, it is necessary to keep collecting malicious HWP documents of various attack types, and examine them to find a better setting.

6. Conclusions

In this paper, to detect malicious actions of HWP files, we designed a convolutional neural network (CNN) that works on byte streams. We extracted byte streams from HWP files and manually identified whether malicious actions were embedded in the streams. By experimental results on our manually annotated 534 HWP files, we demonstrated that the proposed CNN achieves the best performance (e.g., accuracy) amongst comparable machine learning models. We believe that our study will help to prevent the web users from the malicious HWP files by warning them before opening the files. As new contaminated files keep appearing, we need to keep up the study of developing improved detectors. We will continue collecting more malicious HWP files and also investigate better model structures.

Author Contributions: Conceptualization, Y.-S.J.; Investigation, A.R.K.; Methodology, Y.-S.J.; Validation, A.R.K.; Writing—original draft, Y.-S.J.; Writing—review & editing, J.W. and A.R.K.

Funding: This research received no external funding.

Acknowledgments: This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00026, ICT infrastructure protection against intelligent malware threats). This work was supported by the Soonchunhyang University Research Fund. We thank Huy Kang Kim for his comment on the malware exploiting Hangle word processor.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Burges, C.J.C. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.* **1998**, *2*, 121–167. [CrossRef]
2. Breiman, L. Random Forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
3. Medsker, L.; Jain, L.C. *Recurrent Neural Networks: Design and Applications*; CRC Press: Boca Raton, FL, USA, 1999.
4. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
5. Srndic, N.; Laskov, P. Hidost: A static machine-learning-based detector of malicious files. *EURASIP J. Inf. Secur.* **2016**, *22*, 1–20.
6. Cuan, B.; Damien, A.; Delaplace, C.; Valois, M. Malware Detection in PDF Files Using Machine Learning. In Proceedings of the 15th International Conference on Security and Cryptography, Porto, Portugal, 26–28 July 2018; pp. 1–8.
7. Smutz, C.; Stavrou, A. Malicious PDF detection using metadata and structural features. In Proceedings of the 28th Annual Computer Security Applications Conference, Orlando, FL, USA, 3–7 December 2012; pp. 239–248.
8. Li, M.; Liu, Y.; Yu, M.; Li, G.; Wang, Y.; Liu, C. FEPDF: A Robust Feature Extractor for Malicious PDF Detection. In Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Sydney, Australia, 1–4 August 2017; pp. 218–224.
9. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–14.
10. Kim, H.; Jeong, Y.S. Sentiment Classification Using Convolutional Neural Networks. *Appl. Sci.* **2019**, *9*, 2347. [CrossRef]
11. Luong, T.; Pham, H.; Manning, C.D. Effective Approaches to Attention-based Neural Machine Translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 1412–1421.
12. Kolosnjaji, B.; Zarras, A.; Webster, G.; Eckert, C. Deep Learning for Classification of Malware System Call Sequences. In Proceedings of the 29th Australasian Joint Conference on Artificial Intelligence, Hobart, Australia, 5–8 December 2016; pp. 137–149.
13. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
14. Huang, W.; Stokes, J.W. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, San Sebastián, Spain, 7–8 July 2016; pp. 399–418.
15. Zhou, H. Malware Detection with Neural Network Using Combined Features. In Proceedings of the China Cyber Security Annual Conference, Beijing, China, 14–16 August 2018; pp. 448–456.
16. Xiao, F.; Lin, Z.; Sun, Y.; Ma, Y. Malware Detection Based on Deep Learning of Behavior Graphs. *Math. Probl. Eng.* **2019**, *2019*, 8195395. [CrossRef]

17. Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.A. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.* **2010**, *11*, 3371–3408.
18. Raff, E.; Sylvester, J.; Nicholas, C. Learning the PE Header, Malware Detection with Minimal Domain Knowledge. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Dallas, TX, USA, 3 November 2017; pp. 121–132.
19. Saxe, J.; Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In Proceedings of the 10th International Conference on Malicious and Unwanted Software, Fajardo, PR, USA, 20–22 October 2015; pp. 11–20.
20. Le, Q.; Boydell, O.; Namee, B.M.; Scanlon, M. Deep learning at the shallow end: Malware classification for non-domain experts. *Digit. Investig.* **2018**, *26*, S118–S126. [CrossRef]
21. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C. Malware detection by eating a whole EXE. In Proceedings of the Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–3 February 2018; pp. 268–276.
22. Jeong, Y.S.; Woo, J.; Kang, A.R. Malware Detection on Byte Streams of PDF Files Using Convolutional Neural Networks. *Secur. Commun. Netw.* **2019**, *2019*, 8485365. [CrossRef]
23. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
24. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015; pp. 1–15.
25. Cogswell, M.; Ahmed, F.; Girshick, R.B.; Zitnick, L.; Batra, D. Reducing Overfitting in Deep Networks by Decorrelating Representations. In Proceedings of the 4th International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016; pp. 1–12.
26. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In Proceedings of the 2015 IEEE International Conference on Computer Vision, Santiago, Chile, 7–13 December 2015; pp. 1026–1034.
27. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 448–456.