electronics

MDPI

Article

Identifying Symmetric-Key Algorithms Using CNN in Intel Processor Trace

Wooyeol Yang and Yongsu Park *

Department of Computer Science, Hanyang University, Wangshimriro 222, Seongdong-gu, Seoul 04763, Korea; yotop93@naver.com

* Correspondence: yongsu@hanyang.ac.kr

Abstract: Malware and ransomware are often encrypted to protect their own code, making it challenging to apply reverse engineering to analyze them. Recently, various studies have been underway to identify cryptography algorithms in malware or ransomware that use anti-reversing technology via deep-learning technology. In particular, CNNs (convolution neural networks) are deep-learning algorithms with superior performance, as compared to existing machine-learning algorithms in image classification. In the cases of malicious files to which anti-debugging techniques or anti-DBI (dynamic binary instrumentation) techniques are applied, if the traces are extracted using various debuggers or DBI, the traces are cut off due to these techniques. The IPT (Intel processor trace) has the advantage of extracting an accurate trace of a program by bypassing the anti-debugging or anti-DBI technique. This paper presents a novel method by which to identify the symmetric-key algorithms by applying a CNN to the traces extracted from the IPT. The IPT minimally interrupts software execution. First, the trace encrypted by the symmetric-key algorithms is extracted using the IPT. Then it is converted into an image to be an input into the CNN. The experiments were carried out with two different datasets. The first dataset contained traces extracted by different types of symmetric-key algorithms, and the training results were classified into nine classes with 100% accuracy. The second dataset contained traces that included the various bit sizes of the security keys and the block-cipher modes for each type of symmetric-key algorithm. Training results were classified into 36 classes with an accuracy of 70.55%. While previous studies have identified the types of encryption algorithms, this study employed a CNN to identify the number of key bits and the block-cipher modes as well.

Keywords: malware; ransomware; symmetric-key algorithm; Intel processor trace; convolution neural network

Citation: Yang, W.; Park, Y. Identifying Symmetric-Key Algorithms Using CNN in Intel Processor Trace. *Electronics* **2021**, *10*, 2491. https://doi.org/10.3390/electronics10202491

Academic Editor: Marco Vacca

Received: 16 September 2021
Accepted: 8 October 2021
Published: 13 October 2021

1. Introduction

Currently, the number of attacks and the amount of damage caused by malware and ransomware is increasing at a tremendous rate as digital transformation is accelerating globally [1,2]. For example, recently popular ransomware encrypts files on personal PCs and corporate PCs using a cryptography algorithm. It then requires money in exchange for a decryption key. This behavior is recognized as a major threat to individuals and countries [3]. Ransomware can infect a device via emails, web pages, or P2P sites, especially in the case of an encrypted malware or ransomware, which makes it difficult for antivirus programs to detect [4,5]. Since 2016, ransomware has seen a rapid increase in the number of undetected types of software, one of the most dangerous cyberattacks, especially the financial damage caused by it [6].

Encryption prevents the information transmitted from being read or understood by anyone other than the intended party. Therefore, it is often used to hide critical information. However, recently, malicious code makers have often used encryption to hide malicious code [7,8]. According to the application delivery networking company F5 Inc., malware makers encrypt their own malware so that it is not detected by most malware detectors [9].

Furthermore, 71% of malware uses encryption when receiving an attack command or communicating with the control center [10]. Due to this encryption, the incident response team is burdened with identifying the malicious code [10]. To prepare for such an attack, a new strategy is needed to quickly identify and classify samples of malware or ransomware and analyze their behavior. Existing cryptography algorithm detection methods have disadvantages. For example, it is difficult to detect when a new variant of malicious code is released. It is necessary to update the pattern or the signature of the newly created ransomware [11]. Recently, new techniques have been studied to predict and detect cryptography algorithms by learning modified encryption routines using AI (artificial intelligence) technologies.

A recent study concerning the detection of cryptography algorithms using AI is MLCrypto [12]. The study used machine learning to detect TDEA, AES-256, SHA-256, and HMAC-SHA-256. MLCrypto used the naive Bayes classifier, which is efficient in computing, simple to implement, and does not require an adjustment of learning parameters. The experimental results showed a convergence performance of about 50%. CRED [13] used a process- and data-driven approach to detect cryptography algorithms. CRED built an early detection model using both data- and process-driven approaches based on the LSTM algorithm. This allowed the ransomware to be identified before it started an encryption attack. As a result, the model proposed in [13] effectively protected personal and business data. RWguard [14] studied ransomware detection and experimented on samples from 14 widely known ransomware families. The above study monitored all changes (creation, deletion, and writing) in the file and checked for abnormal changes. If CryptoAPI was called during the monitoring, it checked whether the request was normal and then determined if the request was benign or malicious. Additionally, the CPU was used from 0.85% to 1.02%.

Recent studies on deep-learning-based malicious code analysis are in progress, and research results are being produced that classify malicious codes with higher accuracy than existing machine-learning algorithms [15,16]. This research studied the classification of a new symmetric-key algorithm applied with deep learning, paying attention to the fact that the evolving characteristics of malicious codes and ransomware are similar to deep-learning methods that evolve through self-learning. The IPT was used to capture the execution information of all instructions executed by the CPU in a Windows operating system to solve this problem. The CNN deep-learning algorithm was used to identify which kind of symmetric-key algorithm was being applied. Commercial packers such as Themida [17] and Enigma [18] provided anti-reversing techniques and encryption when packing software [19]. The advantage of IPT is that it can bypass various anti-reversing techniques. Moreover, while capturing the execution information of the program, no interruption is applied to the execution information or flow of the program. Existing research on the detection of cryptography algorithms can identify the type of cryptography algorithm being executed. However, sometimes it is impossible to clearly identify the number of key bits and the block-cipher mode used. Even if the cryptography algorithm can be detected, additional time is required to identify the number of key bits and the block-cipher mode.

In [20], they used an Intel processor trace to detect symmetric-key algorithms. They found key generation patterns and routine encryption patterns based on heuristics by extracting a trace using a symmetric-key algorithm. However, the experimental result did not detect the key generation and encryption routine patterns of SEED, IDEA, and SM4 among the symmetric-key algorithms.

In this study, a new approach applying CNN was presented to classify a symmetric-key algorithm with two data sets composed of IPT traces to solve this problem. The first data set was imaged by extracting traces for each type of symmetric-key algorithm (e.g., AES, DES, IDEA, etc.). The second data set was an image of a trace that included the types of symmetric-key algorithms, the numbers of key bits, and the block-cipher modes (AES-128-CBC, AES-192-ECB, and AES-256-CBC). The first data set showed a classification accuracy of 100%, and the second set showed a classification accuracy of 70.55%. The

contribution of this study is as follows. First, it was a new study to detect a symmetric-key algorithm by applying the CNN with an IPT. Second, when analyzing a program in which ransomware or malicious code was inserted, the time and effort required for analysis could be reduced by providing information on the type of symmetric-key algorithm as well as the number of key bits and the block-cipher mode.

The composition of this paper is as follows. Section 2 describes studies related to cryptography algorithm detection. Section 3 describes the proposed technique for detecting symmetric-key algorithms and background knowledge required. Section 4 describes the experiments in this study. Section 5 evaluates the experimental results and performance, and finally, Section 6 presents conclusions and future plans.

2. Related Work

2.1. Detection of Cryptography Algorithms that Do Not Utilize AI

In [21], the study examined how to detect cryptography algorithms using the fact that the parameters of the cryptography algorithm must exist in memory during the execution. It confirmed the existence of a cryptography algorithm by assuming that input and output parameters that match a specific pattern existed in memory and checking whether they could be extracted. A data pattern was defined as a mathematical relationship between the input data and the output data. It consisted of a front-end (Foch) and a back-end (Loch). The front-end extracted a trace of the program using a digital forensic analyzer called Fochs. The back-end determined the cryptography algorithm by extracting useful data based on the traces extracted from the front-end. As a result of the experiment, the known algorithms were successfully discovered, and the parameters of various cryptography algorithms were extracted. Aligot [22] was a study to detect cryptography algorithms in packed programs due to obfuscation or packers using Intel pins. The cryptography algorithm was detected based on the loop data flow of the target program. It detected the algorithm by extracting input and output parameters based on the loop data flow and comparing the parameters of known cryptography algorithms. However, because it used Intel pins, a dynamic analysis tool, it could only identify code that was executed at runtime. In addition, Aligot [22] had the limitation that it could not detect the cryptography algorithms created randomly by malicious code creators.

2.2. Cryptography Algorithm Detection Techniques Using AI

The author of [23] evaluated the performance of the algorithm to classify the DES, IDEA, AES, and RC2 block cryptography algorithms that encrypt in ECB mode. The evaluation algorithm used naive Bayesian, support vector machine, MLP, IBL, AdaBoost M1 with bagging, rotation forest, and decision tree. As a result, the algorithm that used pattern recognition proved to be useful for identifying cryptography algorithms. The study in [24] identified the cryptography algorithm by tracking the number of times a specific instruction was executed using an Intel pin. In this study, SVM, kernel, naive Bayesian, and decision tree were used for evaluation. Among the above algorithms, decision tree showed the best performance overall, but in the case of many classes, the performance was very poor. It did not specify exactly how bad the performance was. In [25], the study focused on detecting ransomware based on the Windows API that was called during encryption. If there was an API that performed encryption among the Windows APIs that were being called, the algorithm stopped the execution and checked if there was a signature in the SR (signature repository). If the signature existed, it was classified as ransomware; otherwise, it was registered as a new signature in the repository. As a result, AUC 0.9930 FPR showed a result of 0.0156. The author of [26] extracted and learned the features of each cryptography algorithm by extracting the binary basic block and loop, the instruction, and the entropy using the dynamic instrumentation through the Intel pin API. The samples used were compiled binary executables. In addition, features were extracted from the trace using a dynamic convolution neural network. Experiments were carried out using the algorithms AES, RC4, Blowfish, MD5, and RSA, and the result was 96%.

Recently, research on detecting malicious codes using CNNs in various classification problems has been actively conducted. Hwang et al. [11] analyzed strings from platform-independent binary data to classify malware on Windows or Linux systems. They extracted strings from binary data, represented them as vectors, and extracted features. Then, they were trained using a DNN and classifications. Based on this result, the endpoint entry was allowed only when the binary was judged to be positive. The experiment showed an accuracy for this method of about 94%. The model can be improved by establishing a security policy through continuous manual analysis and monitoring. Ashik et al. [27] collected various malicious code samples and extracted features such as mnemonics, command opcodes, API calls, and 4-gram mnemonics using dynamic analysis. In addition, these features were learned and evaluated using various machine learning and deep learning. As a result, the accuracy of the CNN in this study was 97.9%. Yan et al. [28] extracted opcode sequences from malicious code files using the IDA decompile tool and generated grayscale images. Then, using a CNN and an LSTM network, the grayscale images and opcode sequences were learned as input, respectively. To optimize detection performance, the output, and metadata functions of the two networks were integrated using a stack ensemble, and the final prediction results were evaluated. The experimental results showed an accuracy of 99.36%.

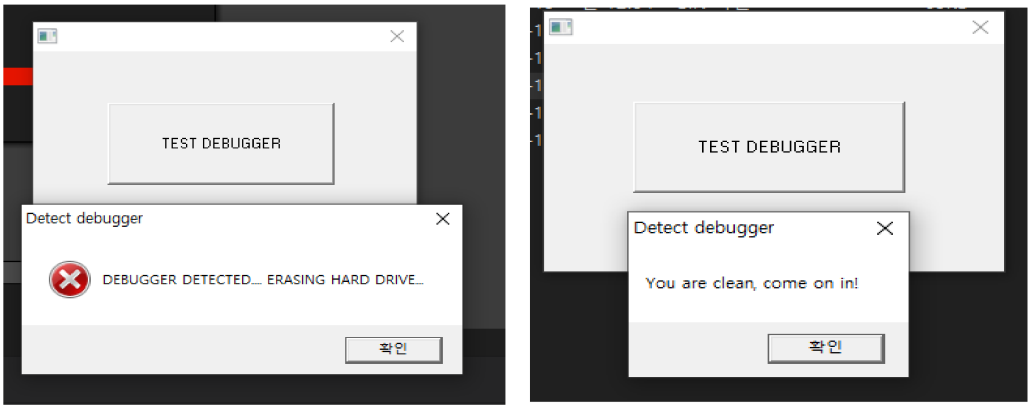
3. Proposed Scheme

3.1. Intel Processor Trace

The IPT is an Intel architecture that captures execution information of running software. The IPT has information such as timing and program flow information (e.g., branch target and branch use) collected in data packets [29]. There are two main categories of data packets extracted by an Intel processor trace. One contains five types of packets (PSB (packet stream boundary), PIP (paging information packet), TSC (time-stamp count), CBR (core bus ratio), and OVF (overflow packet)). The other category has four types: (TNT (taken-not-taken), TIP (target IP), FUP (flow update packet), and MODE (MODE packet)) that have control flow information. Research using these packets is actively being conducted [30–32]. The role of each packet is as follows. The PSB packet is generated at regular intervals and serves as a packet boundary and is the first packet that the decoder finds when starting trace decoding. The PIP packet records modifications to the CR3 register, which sets the physical address of the page directory and the functions related to the page cache. The TSC packet is similar to the CPU's time stamp counter, but it is a time stamp counter for software. The CBR packet stores the bus clock rate. The OVF packet is transmitted when the processor's internal buffer overflows and the packet is deleted. This packet informs the decoder of the loss and allows the decoder to respond to this situation. The TNT packet has direct conditional branch information. The TIP packet records the point of instruction for indirect branches, exceptions, interrupts, and other branches or events. The FUP packet provides source instruction point information when source addresses cannot be determined from binaries or asynchronous events, such as interruptions and exceptions. The MODE packets have detailed information on execution modes such as 16-bit, 32-bit, and 64-bit [29].

"Reversing" is the analysis of software to understand its design or structure. When reversing, various debuggers or DBI (dynamic binary instrumentation) are used. Since the IPT also captures program execution information, it can be used as a reversing tool. "Anti-reversing" refers to a technique that prevents the analysis of software using a reversing tool. In order to protect their own code, malicious code creators apply the anti-reversing technique. One of the characteristics of the IPT is that it can bypass various anti-reversing techniques. It also does not impose any interruptions on the execution information or the flow of the program while capturing the execution information of the program. Figure 1a,b are the results of executing a program to which a simple anti-debugging technique is applied. If you press the "TEST DEBUGGER" button in the center, a message indicates

that the debugger has been detected. Figure 1b shows the execution through IPT. Since IPT bypasses the anti-reversing techniques, the program does not recognize it as a debugger.



(**a**) (**b**)

Figure 1. Example of debugger detection. (**a**) Execution of program with anti-debugging applied. (**b**) Execution through IPT.

Figure 2 shows the assembly code that detects the debugger and accesses the PEB structure by executing the instruction "mov eax, dword ptr fs: [30h]". Furthermore, through "movzx eax, byte ptr ds: [eax + 2]", the value of "BeingDebugged", the second member of the PEB structure, is stored in "eax". BeingDebugged has a value of 1 when being debugged and a value of 0 when not being debugged. Moreover, this value is stored in "eax". Thus, if debugging is performed, the "nop" instruction will be executed after executing the "je label1" instruction. If debugging is not in progress, the "nop" instruction will not be executed after the "je label1" instruction is executed.

```
DetectDebugger macro
      ASSUME fS:nothing

      ; Instead of invoking kernel32!IsDebuggerPresent directly,
      ; we read the PEB structure
      mov eax, dword ptr fs:[30h] ; read PEB
      movzx eax, byte ptr ds:[eax + 2] ; get flag and leaves it in EAX
      cmp eax, 0
      je label1
      nop
label1:
      ; Set variable
      mov dword ptr [intDetectDebug], eax

endm
```

Figure 2. Assembly code to detect the debugger.

Figure 3 shows the trace extracted through the debugger and Windows Intel PT [33]. The trace extracted through Windows Intel PT [33] is on the left, and the trace extracted through the debugger is on the right. As described above, the "eax" value is checked using the "cmp eax 0" instructions and the "jz short loc_401010" instruction is executed. Since IPT bypasses the anti-debugging technique, the "nop" instruction is not executed. However, in the case of the right figure, the "nop" instruction is executed because it is executed through the debugger.

```
401000 mov     eax, large fs:30h
401006 movzx   eax, byte ptr [eax+2]
40100a cmp     eax, 0
40100d jz      short loc_401010
401010 mov     dword_403000, eax
401015 push    0; lpModuleName
401017 call    GetModuleHandleA
4010ca jmp     ds:__imp_GetModuleHandleA
40101c mov     hInstance, eax
401021 call    InitCommonControls
4010e2 jmp     ds:__imp_InitCommonControls
401026 push    0; dwInitParam
401028 push    offset DialogFunc; lpDialogFunc
40102d push    0; hWndParent
40102f push    65h; lpTemplateName
401031 push    hInstance; hInstance
401037 call    DialogBoxParamA
4010d0 jmp     ds:__imp_DialogBoxParamA
40103c push    0; uExitCode
40103e call    ExitProcess
4010c4 jmp     ds:__imp_ExitProcess
```

```
000020B0  401000  mov   eax, large fs:30h          R
000020B0  401006  movzx eax, byte ptr [eax+2]      R
000020B0  40100A  cmp   eax, 0                     PF=0 ZF=0
000020B0  40100D  jz    short loc_401010
000020B0  40100F  nop
000020B0  401010  mov   dword_403000, eax
000020B0  401015  push  0; lpModuleName            R
000020B0  401017  call  GetModuleHandleA           R
000020B0  4010CA  jmp   ds:__imp_GetModuleHandleA  R
000020B0  40101C  mov   hInstance, eax
000020B0  401021  call  InitCommonControls         R
000020B0  4010E2  jmp   ds:__imp_InitCommonControls R
000020B0  401026  push  0; dwInitParam             R
000020B0  401028  push  offset DialogFunc; lpDialogFunc
000020B0  40102D  push  0; hWndParent              R
000020B0  40102F  push  65h; lpTemplateName        R
000020B0  401031  push  hInstance; hInstance       R
000020B0  401037  call  DialogBoxParamA           R
000020B0  4010D0  jmp   ds:__imp_DialogBoxParamA   R
000020B0  401043  Memory layout changed: 259 segments
000020B0  401043  push  ebp                        RSP=00000
```

Figure 3. Trace extracted via IPT and debugger.

3.2. CNN (Convolutional Neural Network)

The convolution neural network is a feedforward neural network inspired by the experiments on the feline visual cortex by David H. Hubel and Torsten Wiesel, as shown in Figure 4. They found that neurons respond only to visual stimuli within a partial range of the field of view and that the receptive fields of these neurons overlap each other to form the entire field of view.

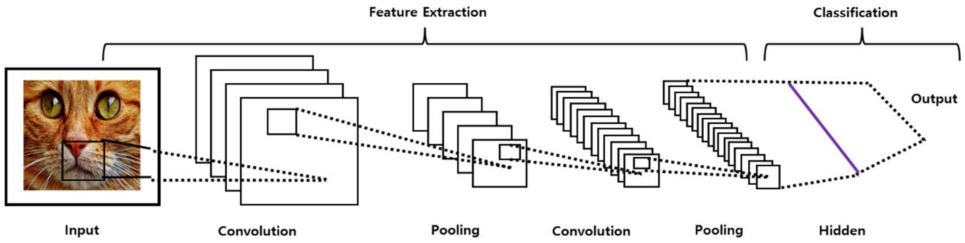


Figure 4. Convolution neural network.

The CNN was first introduced in [34] to process images more effectively using filtering techniques, and then the CNN of the form currently used in deep learning was proposed in [35]. For example, when creating a CNN that recognizes a cat's face, as shown in Figure 4, a convolutional layer is first created by extracting simple features using a filter. Then, a new layer is added that extracts more complexity from these features. This is the principle of extracting high-dimensional features by connecting several convolutional layers and, finally, learning using an error backpropagation neural network as before. Previously, in the field of image recognition, people made these filters themselves, but the biggest advantage of CNNs is that they create filters through learning. The CNN is the most popular algorithm among deep-learning algorithms because each element of the filter is trained appropriately for data processing and has excellent performance in classification.

The CNNs perform very well, especially in image processing problems. For this reason, this work solves the problem of identifying symmetric-key algorithms in traces extracted from IPTs by applying CNNs. Each symmetric-key algorithm classification problem was converted into an image classification problem and classified by the symmetric-key algorithm by learning using a CNN. Table 1 in Appendix A shows the parameters of the CNN in our experiments.

3.3. System Overview

Figure 5 shows an overview of the systems proposed in this study. First, we extracted the encryption routine trace using the Windows Intel PT [33] from computer A in a Windows environment. This study extracted 550 traces from 500 training sets and 50 test

sets for each symmetric-key algorithm. We moved the extracted traces to computer B in a Windows environment. Using the Windows Intel PT [33], the .bin file and the .txt file appeared, and the Python code that converts binary to an image was operated on computer B and trained the image using the CNN.

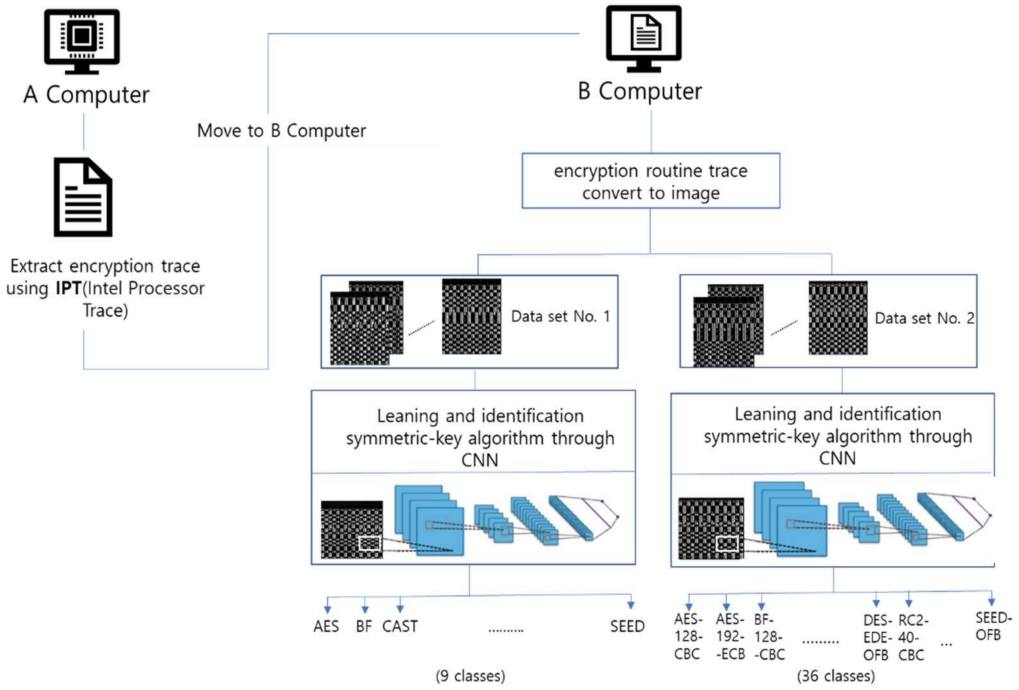


Figure 5. Overview of the proposed scheme.

3.3.1. Extracting Encryption Trace

We downloaded the OpenSSL library to computer A and extracted the encryption routine trace using the Windows Intel PT [33]. Computer B converted all traces to images in order to learn using the CNN. The way to extract traces from computer A was to use the Windows Intel PT [33], which was downloaded from GitHub, Inc.

3.3.2. Symmetric-Key Algorithm Trace Imagination

The types of symmetric-key algorithms are AES, BF, CAST, DES, DES3, IDEA, RC2, RC4, and SEED, comprising nine types altogether. Although there are various types of ransomware, they generally have one thing in common: they perform encryption. Even when a new ransomware is created, the ransomware has the common feature of encryption, so in this study, the most widely used open-source library OpenSSL was downloaded, and the encryption trace was extracted and imaged. Figure 6 is a depiction of the encryption trace as a grayscale image. If the trace was extracted using a Windows Intel PT [33], the .bin file was output. This .bin file is shown in Figure 6 and is imaged using the Python code that images this type of .bin file. The size of the image depends on the size of the .bin file. Figure 7 is a Python pseudocode that images a .bin file. This code reads the .bin file by 8-bits and adds it as a unicode value to the "binaryValue" variable. We repeat this process until the end of the .bin file to add the value of "binaryValue" and save the size. We set the width value according to the size, and when the width value was determined, the height value was also calculated and saved. Finally, when the width and height values were set, an L option (gray-scale mode) line was given and created a gray-scale image.

Figure 6. The process of converting .bin file to image.

```
binaryValue = []
if exist bin file
        data = file.read(1) //read file 8bit at a time
        while data != b""
                binaryValue.append(ord(data)) //Adds the Unicode value of the data character to binaryValue.
                data = file.read(1)
size = len(binaryValue)
if size < 10240
        width = 64
elif 10240 <= size <= 10240 * 3
        width = 128
                .
                .
                .
elif 10240 * 50 <= size <= 10240 * 100
        width = 768
else :
        width = 1024
height = int(size / width) + 1
image = Image.new('L', (width, height)) //L option is gray-scale mode
```

Figure 7. Pseudocode that converts binary to image.

4. Experiment

According to a recent survey, 83% of PCs worldwide operate in the Windows environment, and malware research is carried out mainly on Windows operating systems. We used the IPT in a Windows environment, but it can also be used in the Linux environment [36]. Because this study detects the cryptography algorithm by learning the encryption routine in a Windows operating system, it makes a great contribution to the practical application of identifying ransomware.

4.1. Experiment Environment

In this study, we used two computers, one (computer A) to extract the trace, and the other (computer B) to learn using the CNN. We used two computers instead of one for the following reason. The GPU we used had a GTX 1660 Super graphics card, but the driver to support this graphics card was not compatible with the Windows version that uses Windows Intel PT [33]. Windows Intel PT [33] runs on Windows version 1607, build number 14393.0, and, therefore, must be equipped with an Intel Skylake CPU. However, among the drivers for the GTX 1660 Super, none supported Windows version 1607, so we used two computers. The experimental environment of this study is shown in Table 1. Computer A had Windows 10 version 1607 with an Intel CPU i5-6600. Computer A extracted the encryption-routine trace of the OpenSSL symmetric-key algorithm. The number of samples used was 550 by combining 500 training sets and 50 test sets for each symmetric-key algorithm. Furthermore, the environment of computer B training the above samples was the Windows 10 20H2 version with the GTX 1660 Super graphics card, Intel i7-8700 CPU, and 32GB RAM. Table 1 in Appendix A showed the parameters of the CNN in our experiments.

Table 1. Experiment environment.

Computer Name	CPU	RAM	GPU
A	Intel i5-6600	8 GB	none
B	Intel i7-8700	32 GB	gtx 1660 super

4.2. Data Set

In this study, traces extracted from the Windows Intel PT [33] were used with two data sets. The composition of the data set for each symmetric-key algorithm was shown in Table 2. It indicated the number of training data sets and the number of test data sets used for each symmetric-key algorithm. The first data set consisted of nine symmetric-key algorithms, 500 training data sets per algorithm, and 50 test data sets. The second data set consisted of 36 symmetric-key algorithms with the same the number of training and test sets as first data set. Figure 8 is an image of nine symmetric-key algorithms that make up the first data set. Figure 9 is an image of 36 symmetric-key algorithms that make up the second data set. For each set, the CNN training was performed.

Table 2. Data Set.

Data Set No.	Symmetric-Key Algorithm	Number of Data per Algorithm	
1	AES, BF, CAST, DES, DES3, IDEA, RC2, RC4, SEED	Training Set Test Set Total	500 50 4950
2	AES-128-CBC, AES-128-ECB, AES-192-CBC, AES-192-ECB, AES-256-CBC, AES-256-ECB, BF-CBC, BF-CFB, BF-ECB, BF-OFB, CAST-CBC, CAST5-CFB, CAST5-ECB, CAST5-OFB, DES-CBC, DES-CFB, DES-ECB, DES-EDE-CBC, DES-EDE-CFB, DES-EDE-OFB, DES-OFB, IDEA-CBC, IDEA-CFB, IDEA-ECB, IDEA-OFB, RC2-40-CBC, RC2-64-CBC, RC2-CBC, RC2-CFB, RC2-ECB, RC2-OFB, RC4-40, SEED-CBC, SEED-CFB, SEED-ECB, SEED-OFB	Training Set Test Set Total	500 50 19,800

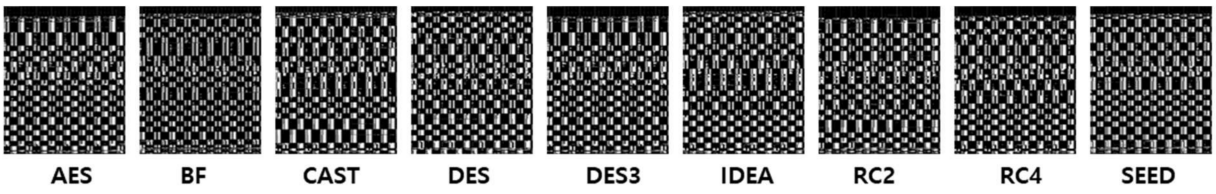


Figure 8. Image of symmetric-key algorithm in the first data set.

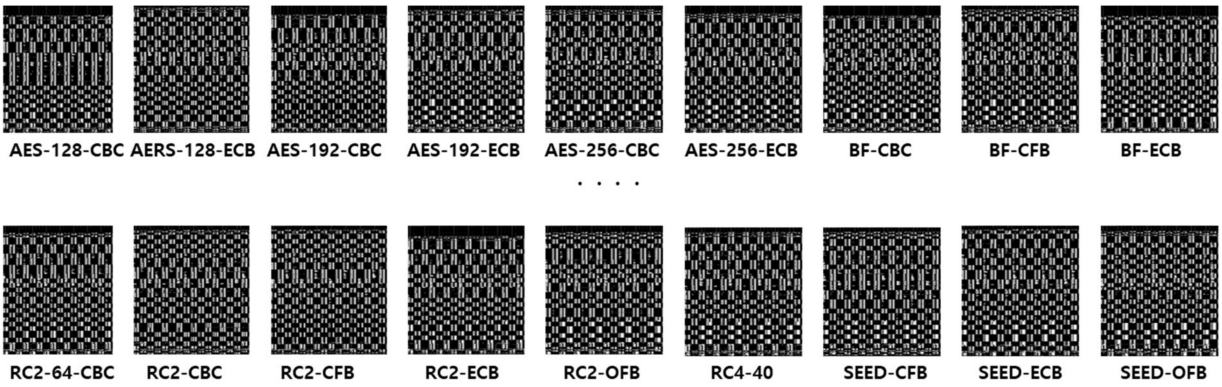


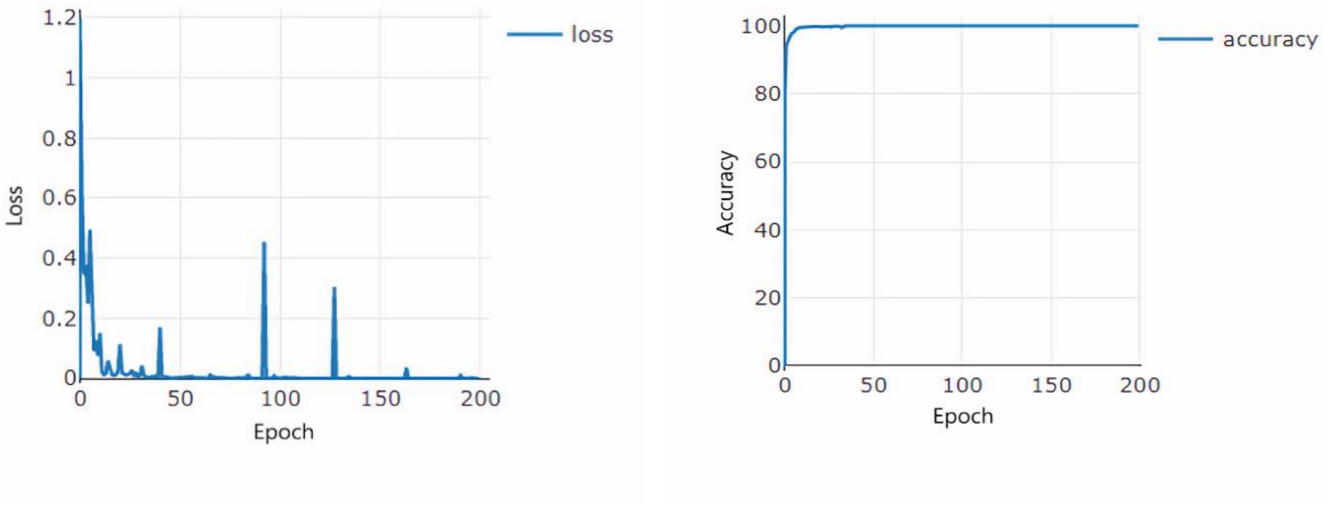
Figure 9. Image of symmetric-key algorithm in the second data set.

5. Evaluation

This study verified the possibility of detecting symmetric-key algorithms by combining the IPT and a CNN. The performance showed 100% accuracy in identifying the types of symmetric-key algorithms, and the identification accuracy of the key sizes and

the block-cipher modes showed 70.55% accuracy. The performance of recent (first data set) studies [23,26] on the symmetric-key algorithm was 92–96%, but our study showed 100% accuracy, which meant there was almost no misidentification. Furthermore, while previous studies [24,26] identified only the type of symmetric-key algorithm used, this study showed the results of identifying the number of key bits and the block-cipher modes with 70.55% accuracy.

Figure 10a shows the loss per epoch while the model was training on the first data set. A total of 200 epochs were performed; the x-axis represents the epoch, and the y-axis represents the loss. As the epoch progresses, the loss converges to 0. Figure 10b is a graph showing the accuracy of each epoch while the first data set model was training. Again, the x-axis represents the number of epochs, and the first epoch learning showed about 70.55% accuracy, but as the epochs progressed, the accuracy finally showed 100% performance.



(**a**) (**b**)

Figure 10. Performance of first data set. (**a**) loss (%), (**b**) accuracy (%).

Figure 11 is a graph showing the accuracy per test sample during the test of the first data set. The x-axis represents the number of test samples, and the y-axis represents the accuracy. It can be converged to 100%, which is similar to the accuracy in the training process. This showed that symmetric-key algorithms can be classified correctly.

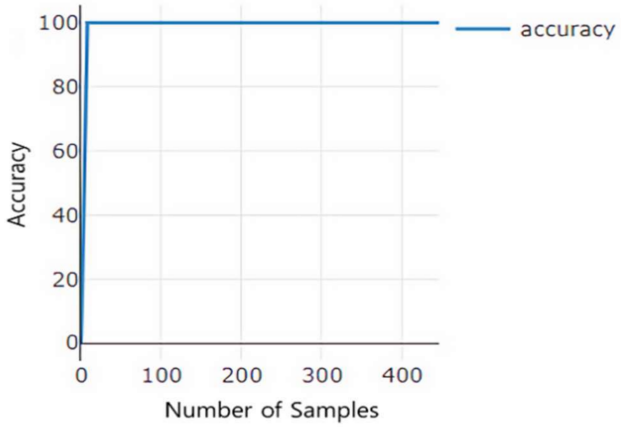
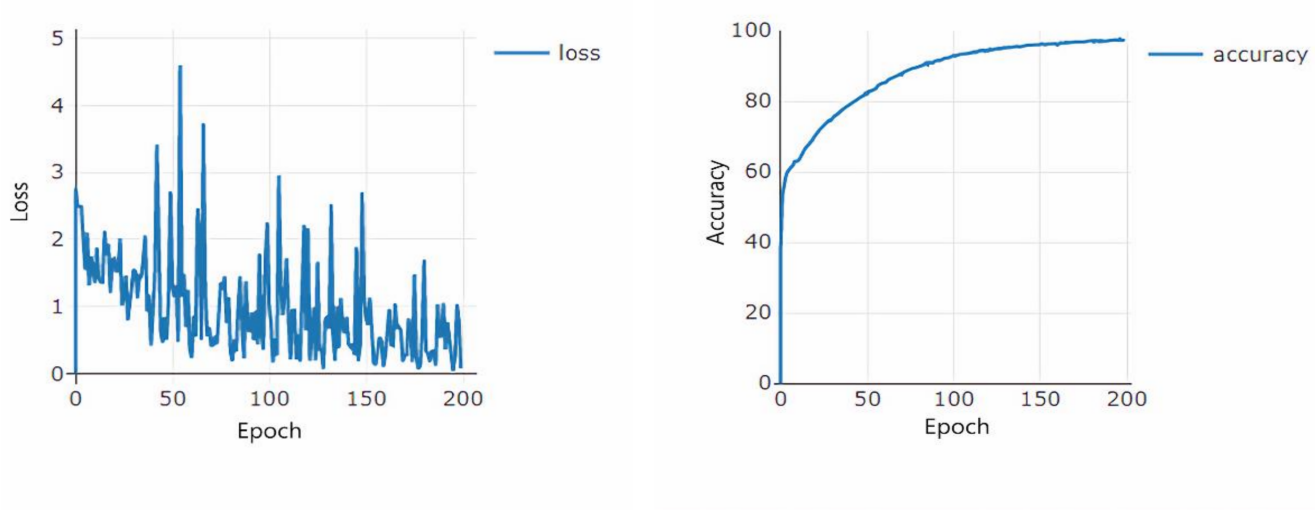


Figure 11. Accuracy of test of the first data set.

Figure 12a is a graph showing the loss per epoch during the training of the second data set. The x-axis represents the number of epochs, and the y-axis represents the loss. A total of 200 epochs were performed, and there was more loss than in the first data set.

This was assumed because the number of samples was larger than that of the first data set. However, the error rate of the loss gradually decreased as it converged to zero as the epochs progressed. Figure 12b is a graph showing the accuracy measured while training the second data set. The x-axis represents the number of epochs, and the y-axis represents the accuracy. Since the number of samples to be trained was large, the convergence to 100% occurred after 150 epochs, as compared to the first data set.



(**a**) (**b**)

Figure 12. Performance of the second data set. (**a**) loss (%), (**b**) accuracy (%).

Figure 13 is a graph that represents the accuracy of the second data set. The x-axis represents the test sample, and the y-axis represents the accuracy. In the training process, the accuracy converged to 100%, but the test process showed an accuracy of about 71%. This was presumed to show lower accuracy than the first data set as the number of bits for each type of symmetric-key algorithm and the sample including the block-cipher modes were trained and tested. In [23], the accuracy of their scheme can be lowered by 30–50% when they used various key sizes, encryption modes, and numbers of keys. From this, we deduced that the accuracy of previous works for various key lengths and encryption modes that were similar to the second data set was not very high.

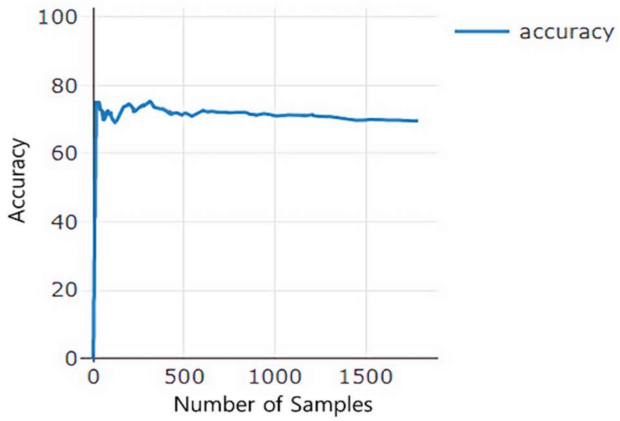


Figure 13. Accuracy of second test data set.

To improve this, we planned to solve the problem by increasing the number of training data sets so that the CNN could determine the number of bits by the type of symmetric-key algorithm and the block-cipher mode involved, or by making the layer deeper so that the difference could be accurately learned.

6. Discussion and Conclusions

Deep learning is an artificial intelligence technology that is used to solve classification problems in various fields. In particular, the CNN is in the spotlight as an algorithm with the best classification power out of many deep-learning algorithms. This study presented a new approach to classify symmetric-key algorithms by applying the CNN to traces extracted by the IPT. This study was based on the fact that the characteristics of newly evolving malware are similar to deep-learning approaches that have evolved through the training of learning data. Existing research could only identify what kind of cryptography algorithm was present when analyzing ransomware or malicious code. However, in this study, identifying and learning the traces extracted through the IPT not only defines what kind of algorithm it is, but also provides information on the number of key bits and the block-cipher mode of the symmetric-key algorithm. Using the IPT, our scheme minimally interrupts software execution.

Experiments were conducted on two different data sets. The CNN training results for the first data set were identified by nine types of symmetric-key algorithms with 100% accuracy. Cryptoknight [26] showed 96% accuracy in determining the type of cryptography algorithm. On the other hand, this study showed a 3% improvement in performance. This can reduce the time and effort required for analysis. In the second data set, 36 types of symmetric-key algorithms were identified with 70.55% accuracy by learning traces that included not only the types of symmetric-key encryption algorithms, but also the number of key bits and block-cipher modes for each type of symmetric-key algorithm. In the future, we will conduct research that can show a higher accuracy by considering how to reconstruct the trace or adjust the convolutional layer of the CNN.

Author Contributions: Conceptualization, W.Y. and Y.P.; methodology, W.Y. and Y.P.; software, W.Y.; validation, W.Y. and Y.P.; formal analysis, W.Y.; investigation, Y.P.; resources, W.Y.; data curation, W.Y.; writing—original draft preparation, W.Y.; writing—review and editing, W.Y. and Y.P.; visualization, W.Y. and Y.P; supervision, Y.P.; project administration, Y.P.; funding acquisition, Y.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT), grant number 2020R1F1A1048443.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A

Table 1 shows the parameters of CNN in our experiments.

Table 1. CNN parameter (Number of Classes: 9 or 36).

Layer	Input Channel	Filter	Output Channel or Output Size	Stride	Activation Function	Parameter Number
Convolution Layer 1	3	(3, 3)	16	1	Leaky ReLU	432
Max Pooling Layer 1	X	(2, 2)	X	2	X	0
Convolution Layer 2	16	(3, 3)	32	1	Leaky ReLU	4608
Max Pooling Layer 2	X	(2, 2)	X	2	X	0
Convolution Layer 3	32	(3, 3)	64	1	Leaky ReLU	18,432
Max Pooling Layer 3	X	(2, 2)	X	2	X	0
Convolution Layer 4	64	(3, 3)	128	1	Leaky ReLU	73,728
Max Pooling Layer 4	X	(2, 2)	X	2	X	0
Convolution Layer 5	128	(3, 3)	256	1	Leaky ReLU	294,912
Max Pooling Layer 5	X	(2, 2)	X	2	X	0
Convolution Layer 6	256	(3, 3)	Number of Classes	1	Leaky ReLU	(2304) · (Number of Classes)
Batch Normalization	Number of Classes	X	X	X	X	0
Global Average Pooling Layer	X	X	(1, 1) (: output size)	X	X	0

References

1. Orman, H. Evil offspring-ransomware and crypto technology. *IEEE Internet Comput.* **2016**, *20*, 89–94. [CrossRef]
2. Mohurle, S.; Patil, M. A brief study of Wannacry threat: Ransomware attack. *Int. J. Adv. Res. Comput.* **2017**, *8*, 1938–1940.
3. Sharma, P.; Zawar, S.; Patil, S.B. Ransomware analysis: Internet of things (iot) security issues, challenges and open problems inthe context of worldwide scenario of security of systems and malware attacks. *Int. J. Innov. Res. Sci. Eng.* **2016**, *2*, 177–184.
4. Rieck, K.; Holz, T.; Willems, C.; Düssel, P.; Laskov, P. Learning and Classification of Malware Behavior. In Proceedings of the 2008 International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Paris, France, 10–11 July 2008.
5. Filiol, E. Malicious cryptography techniques for unreversable (malicious or not) binaries. *arXiv* **2010**, arXiv:1009.4000.
6. SONICWALL Annual Threat Report. Available online: https://bluekarmasecurity.net/wp-content/uploads/2017/06/SonicWall-2017-Annual-Threat-Report.pdf (accessed on 30 August 2021).
7. Kansagra, D.; Jumhar, M.; Jha, D. Ransomware: A Threat to Cyber security. *Int. J. Comput. Sci. Commun.* **2016**, *7*, 224–227.
8. SecureOPS. Available online: https://secureops.com/blog/encryption-and-malware-2/ (accessed on 30 August 2021).
9. Detect Encrypted Malwar. Available online: https://www.f5.com/resources/library/encrypted-threats/ssl-visibility/encrypted-malware#spa-1 (accessed on 30 August 2021).
10. Anderson, B.; McGrew, D. Identifying Encrypted Malware Traffic with Contextual Flow Data. In Proceedings of the ACM Workshop on Artificial Intelligence and Security, Denver, CO, USA, 13 November 2016; pp. 35–46.
11. Hwang, C.; Hwang, J.; Kwak, J.; Lee, T. Platform-Independent Malware Analysis Applicable to Windows and Linux Environments. *Electronics* **2020**, *9*, 793. [CrossRef]
12. Brunetta, C.; Picazo-Sanchez, P. Modelling cryptographic distinguishers using machine learning. *Cryptogr. Eng.* **2021**. [CrossRef]
13. Alqahtani, A.; Gazzan, M.; Sheldon, F. A proposed Crypto-Ransomware Early Detection(CRED) Model using an Integrated Deep Learning and Vector Space Model Approach. In Proceedings of the 2020 10th Annual Computing and Communication Workshop and Conference, Las Vegas, NV, USA, 6–8 January 2020.
14. Mehnaz, S.; Mudgerikar, A.; Bertino, E. RWGuard: A Real-Time Detection System Against Cryptographic Ransomware. In Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, Heraklion, Crete, Greece, 10–12 September 2018.
15. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.; Wang, Y.; Iqbal, F. Malware Classification with Deep Convolutional Neural Networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018.
16. Kimmell, J.C.; Abdelsalam, M.; Gupta, M. Analyzing Machine Learning Approaches for Online Malware Detection in Cloud. *arXiv* **2021**, arXiv:2105.09268.
17. Themida. Available online: https://www.oreans.com/Themida.php (accessed on 30 August 2021).
18. Enigma Protector. Available online: https://enigmaprotector.com/ (accessed on 30 August 2021).
19. Omachi, R.; Murakami, Y. Packer Identification Method for Multi-layer Executables with k-Nearest Neighbor of Entropies. In Proceedings of the 2020 International Symposium on Information Theory and Its Applications (ISITA), Kapolei, HI, USA, 24–27 October 2020.
20. Park, J.; Park, Y. Symmetric-Key Cryptographic Routine Detection in Anti-Reverse Engineered Binaries Using Hardware Tracing. *Electronics* **2020**, *9*, 957. [CrossRef]
21. Zhao, R.; Gu, D.; Li, J.; Yu, R. Detection and Analysis of Cryptographic Data Inside Software. In Proceedings of the ISC: International Conference on Information Security, Xi'an, China, 26–29 October 2011.
22. Calvet, J.; Fernandez, J.M.; Marion, J.Y. Aligot: Cryptographic Function Identification in Obfuscated Binary Programs. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012.
23. Sharif, S.O.; Mansoor, S.P. Performance Evaluation of Classifiers used for Identification of Encryption Algorithm. *ACEEE Int. J. Netw. Secur.* **2011**, *2*, 42–45.
24. Hosfelt, D.D. Automated detection and classification of cryptographic algorithms in binary programs through machine learning. *arXiv* **2015**, arXiv:1503.01186.
25. Kok, S.H.; Abdullah, A.; Jhanjhi, N.; Supramaniam, M. Prevention of Crypto-Ransomware Using a Pre-Encryption Detection Algorithm. *Computers* **2019**, *9*, 79. [CrossRef]
26. Hill, G.; Bellekens, X. CryptoKnight: Generating and Modelling Compiled Cryptographic Primitives. *Information* **2018**, *9*, 231. [CrossRef]
27. Ashik, M.; Jyothish, A.; Anandaram, S.; Vinod, P.; Mercaldo, F.; Martinelli, F.; Santone, A. Detection of Malicious Software by Analyzing Distinct Artifacts Using Machine Learning and Deep Learning Algorithms. *Electronics* **2021**, *10*, 1694. [CrossRef]
28. Yan, J.; Qi, Y.; Rao, Q. Detecting Malware with an Ensemble Method Based on Deep Neural Network. *Secur. Commun. Netw.* **2018**, *2018*, 7247095. [CrossRef]
29. Intel Developer Zone. Available online: https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html (accessed on 28 June 2021).
30. Ge, X.; Cui, W.; Jaeger, T. GRIFFIN: Guarding Control Flows Using Intel Processor Trace. *ACM Sigplan Not.* **2017**, *52*, 585–598. [CrossRef]
31. Chen, L.; Sultana, S.; Sahita, R. HeNet: A Deep Learning Approach on Intel Processor Trace for Effective Exploit Detection. In Proceedings of the 2018 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018.

32. Gu, Y.; Zhao, Q.; Zhang, Y.; Lin, Z. PT-CFI: Transparent Backward-Edge Control Flow Violation Detection Using Intel Processor Trace. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AR, USA, 22–24 March 2017.
33. WindowsIntelPT. Available online: https://github.com/intelpt/WindowsIntelPT (accessed on 30 August 2021).
34. Lecun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Backpropagation applied to handwritten zip code recognition. *IEEE Neural Comput.* **1989**, *1*, 541–551. [CrossRef]
35. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. GradientBased Learning Applied to Document Recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
36. Kleen, A.; Strong, B. Intel® Processor Trace on Linux. Tracing Summit. 2015, pp. 1–18. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.735.3516&rep=rep1&type=pdf (accessed on 30 August 2021).