International Journal of Security and Networks

ISSN online: 1747-8413 - ISSN print: 1747-8405
https://www.inderscience.com/ijsn

A hybrid malware analysis approach for identifying process-injection malware based on machine learning

Chia-Mei Chen, Ze-Yu Lin, Ya-Hui Ou, Jiunn-Wu Lin

DOI: 10.1504/IJSN.2024.10062787

Article History:

Received:	13 July 2021
Accepted:	19 July 2021
Published online:	12 March 2024

A hybrid malware analysis approach for identifying process-injection malware based on machine learning

Chia-Mei Chen* and Ze-Yu Lin

Department of Information Management,
National Sun Yat-sen University, Taiwan
Email: cchen@mail.nsysu.edu.tw
Email: fm061j42k7bp6@gmail.com
*Corresponding author

Ya-Hui Ou

General Competency Center,
National Penghu University of Science and Technology, Taiwan
Email: yhou0603@gmail.com

Jiunn-Wu Lin

Kaohsiung Veterans General Hospital, Taiwan
Email: jiunnwu@vghks.gov.tw

Abstract: Advanced persistent threat (APT) attacks take place every day, utilising stealthy and customised malware to disrupt the service or sabotage the network. Such advanced malware may subvert the defence mechanism by abusing process injection techniques provided by operating system and injecting malicious code into a benign process. Some process injection techniques may be identified by static analysis, but some can only be discovered at run time execution. This study adopts deep learning models and two malware analysis approaches to detect process injection malware. By applying transfer learning, this study proposes a CNN-based detection model with the features selected from static and dynamic analysis to identify process-injection malware. The experimental results demonstrate that the proposed method could detect process-injection malware efficiently as well as unknown malware.

Keywords: malware detection; process injection; machine learning.

Reference to this paper should be made as follows: Chen, C-M., Lin, Z-Y., Ou, Y-H. and Lin, J-W. (2024) 'A hybrid malware analysis approach for identifying process-injection malware based on machine learning', *Int. J. Security and Networks*, Vol. 19, No. 1, pp.20–30.

Biographical notes: Chia-Mei Chen has joined in the Department of Information Management, National Sun Yat-Sen University since 1996. She was the Section Chef of Network Division and Deputy Director, Office of Library and Information Services in 2009–2011. She had served as a coordinator of Taiwan Computer Emergency Response Team/Coordination Center (TWCERT/CC) during 1998 to 2013 and then serves as a consultant. Based on her expertise, she established Taiwan Academic Network Computer Emergency Response Team (TACERT) in 2009. She is the Deputy Chair of TWISC@NCKU, a branch of Taiwan Information Security Center. She continues working for the network security society. Her current research interests include anomaly detection, malware analysis, network security, and cloud computing.

Ze-Yu Lin received his MS degree in Department of Information Management from the Management College, National Sun Yet-sen University, Taiwan in 2018.

Ya-Hui Ou has joined in the National Penghu University of Science and Technology as an assistant professor in 2020. She received her MS degree in Department of Information Management from the College of Electrical and Information Engineering, I-Shou University, in 2009, and PhD in Department of Information Management from the National Sun Yat-sen University of Taiwan in 2017. Her research interests include network security and statistical analysis.

Jiunn-Wu Lin has joined the Kaohsiung Veterans General Hospital as a Network and Information Security Engineer since 2003. He received his MS degree in the Department of information Management from National Chi Nan University.in 2003 and PhD in the Department of Information Management from National Sun Yat-sen University of Taiwan in 2020.

1 Introduction

Information technologies and networks offer unprecedented opportunities for businesses, industries, and governments, while the explosive expansion of using information technologies also attracts cybercrimes. Even though networks are protected with defence mechanisms such as anti-virus, spam filter, intrusion detection system, and firewall, organisations still suffer from cyberattacks. Cybercrimes have gone globally and become sophisticated and stealthy.

Malware is a vehicle to compromise victim machines, steal sensitive data, disrupt services, or sabotage the network. Malware refers to malicious programs intentionally developed to infiltrate or compromise systems without the user's consent. In order to bypass security safeguard, some malware developers make use of process injection techniques that injects and executes malicious code in the address space of a legitimate process.

Process injection is a widespread defence evasion technique often employed within malware and fileless attacks (Hosseini, 2017), which entails running custom malicious code within the address space of a legitimate process. Such process-injection malware has caused disruptive targeted attacks (Gorelik, 2017; Kindlund, 2013), for example, Meterpreter DLL Injection, Poison Ivy, and Cyber Gate. Poison Ivy injects malware into the user's default browser, such as explorer.exe, and plays a key role in high-profile advanced persistent threat (APT) attacks, like RSA SecureID and the Nitro assault against chemical makers, government offices, defence firms, and human-rights groups.

Legitimate software adopts process injection as well for performing certain functionalities. For example, a debugger employs it to hook into a program so that the programmer can troubleshoot the program. Anti-virus software is another instance that injects itself into a browser to examine the browser's behaviour and website content (Angelystor, 2020).

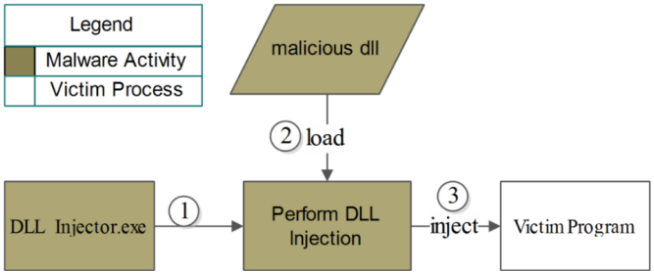
In general, programs invoke Windows API function calls to communicate with the operating system, so API function calls define program behaviours and could represent as features representing malware behaviours (Ngo et al., 2020). For illustration, API calls CreateRemoteThread, LoadLibrary, and WriteProcessMemory are invoked by process-injection malware to create a remote thread, load a DLL, and write it to the memory space of another process.

Malware developers produce new and variants of malware constantly and rapidly. The study (Rosenquist, 2015; Cesare et al., 2013) indicated that malware increased by the growth rate of 50% each year and 88% are variants of a malware sample. They apply stealthy techniques to customise malware for the target environments and adopt evasive techniques to circumvent detection mechanisms. Furthermore, tools are available for performing anti-virus evasion tasks (Srinivas, 2021).

Based on MITRE ATT&CK (Mitre, 2021), there are several techniques for implementing process-injection malware. It is hard to detect as well as hard to mitigate with a preventive control since it is based on the abuse of an operating system design feature. This attack technique has been used by APT groups including APT3 and Cobalt Group. These attacks leverage known vulnerabilities and phishing campaigns to gain entry, run code in the memory space of a benign process. This attack technique is able to trick anti-virus software, as it is difficult to distinguish between legitimate and malicious program (Minerva, 2021). Infected programs can be disseminated through various channels such as email attachments or web links.

A process-injection malware attack scenario is illustrated in Figure 1. Once a user opens the attachment or clicks the web link, the malicious code injects into the address space of a legitimate process. Many tools are available for generating such customised malware (ProcessInjectionTool, 2019; InjectProc, 2019; WinPIT, 2018; Injection, 2019; dirty-needle, 2017; DLL-Inj3cti0n, 2015; ProcessInjection, 2020), and a study (Granneman, 2013) demonstrated that such stealthy attack technique can bypass anti-virus detection easily.

Figure 1 An illustration of process injection (see online version for colours)



Based on the real security incident case studies (TaNet, 2021), in order to retain control over victim machines, adversaries may apply a process injection technique to inject malicious code into a normal process, such as svchost (Kaspersky, 2021; Chang, 2016) or explorer (Kindlund, 2013). A lesson learned from the literature review and incident investigations is that an attacker could adopt the aforementioned evasion technique to craft malware customised for the target organisation and successfully subverted its defence mechanism. Therefore, detecting process-injection malware efficiently is critical to defending against APT attacks.

Two approaches, static and dynamic analysis, are commonly used for detecting malware. Static analysis detects malware without executing the code, disassembles binary executables, extracts signatures or features, and employs a detection model to identify malware. Most static analysis methods detect misbehaviours by a set of sensitive API functions. However, some process injection techniques inject DLLs only during run time, for example fileless malware that injects malicious code without dropping itself in disk space (Gorelik, 2017), and would not be detected by static analysis.

Dynamic analysis is an effective approach to overcome the above drawback. Most past works focused on detecting

malware (Angelystor, 2020; Sihwail et al., 2018) but rarely addressed detecting process-injection malware. To our best knowledge, this study is the first attempt for developing a hybrid malware analysis approach to identifying process-injection malware, which cascades static analysis and then dynamic hooking technique to identify injected code. The first stage of static analysis examines if executable files invoke the sensitive API calls; the second stage applies dynamic hooking to capture injected DLLs and employs a CNN classification model to identify malicious DLLs.

Inspired by the past research that a neural network detection model is efficient on malware detection (Kozachok and Kozachok, 2018) and that convolutional neural network (CNN) models are efficient on imaging recognition, this study applies transfer learning to transform an executable file into an image and develops a CNN-based detection model to identify process-injection malware.

The rest of the paper is structured as follows. Section 2 summarises the background knowledge of process injection techniques and the relevant related work. Section 3 describes the proposed methodology. Section 4 presents the performance evaluation, followed by the concluding remarks and future work in Section 5.

2 Related work

2.1 Process injection

Windows system is one of the most common desktop operating systems with over 77% market shares (Statista, 2021). Process injection is a widespread detection evasion approach to inject malicious code into another process by applying legitimate functionalities provided by the operating system. Table 1 (Hosseini, 2017; Antoniewicz, 2013) summarises the common process injection techniques and applied API functions.

2.2 Malware detection

A malware analysis guide (Sikorski and Honig, 2012) suggested the attributes from portable executable (PE) file, such as header, import table address, strings, and functions, are useful features for analysing malware. PE is a file format for executables and DLLs in Windows systems. Rezaei et al. (2016) extracted opcode strings from the code section of the executable file and applied Misha similarity and edit distance to identify malware. One of the process injection techniques is to inject malicious code into the holes (no-op areas) of a target PE file. A study (Chang, 2016) adopted static analysis to detection such PE-infection malware, it extracts features from PE header and section addresses and identifies malware by distance discrepancy between the locations of the DLLs.

Dynamic analysis examines code by executing it in a controlled environment (aka sandbox) and observes the actions, where the actions can be observed at different levels, such as instruction level to system level. Polino et al. (2015) proposed a dynamic analysis system that reports a descriptive summary for analysts querying the behaviours including invoked API function calls and data-flow dependencies. However, it requires security professions to verify the correctness of the generated semantic tagging.

Table 1 Process injection techniques and applied APIs

Process injection	*API*
DLL injection	CreateToolhelp32Snapshot, Process32First, Process32Next, OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, NtCreateThreadEx, RtlCreateUserThread, LoadLibrary
PE injection	CreateRemoteThread, VirtualAllocEx, WriteProcessMemory
Process hollowing	CreateProcess, ZwUnmapViewOfSection, NtUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory, SetThreadContext, ResumeThread
Thread execution hijacking	CreateToolhelp32Snapshot, Thread32First, OpenThread, SuspendThread, VirtualAllocEx, WriteProcessMemory, SetThreadContext, (LoadLibrary)
Hook injection	LoadLibrary, GetProcAddress, CreateToolhelp32Snapshot, Thread32Next, SetWindowsHookEx
Injection via registry modification	RegCreateKeyEx, RegSetValueEx
Asynchronous procedure call injection	OpenThread, QueueUserAPC, (LoadLibraryA)
Extra window memory injection	GetWindowLong, SetWindowLong, SendNotifyMessage, (NTMapViewOfSection)

Given the fact that static and dynamic analysis have their pros and cons. Researchers combine both to improve the performance. Choi et al. (2012) applied open source tools to extract features from static and dynamic analysis and adopt a machine learning classification model to identify malware. However, they proposed the idea with the lack of experiments. Ye et al. (2017) concluded that the detection process is divided into two stages: feature extraction and classification/clustering and summarised that static analysis features include DLLs, APIs, opcode sequences, control flow graph, and strings and dynamic ones include memory, network activities, API call sequence, and system calls.

A study (Sun, 2020) built a signature-like profile for each benign program that records its DNS query behavior and detected malware-injected processes whose DNS activities deviate from the benign program. Another study (Hăjmăşan et al., 2017) implemented a minifilter driver to monitor kernel-mode actions of the running processes and applied a scoring mechanism to capture suspicious processes. The user needs to make decisions whether the captured actions are to be allowed or blocked.

A study (Ijaz et al., 2019) evaluated the detection efficiency of static and dynamic analysis features with thousands of malicious and benign executable files and concluded that dynamic malware analysis is not effective due to malware's stealthy and intelligent behaviours. The dynamic analysis cannot analyse a target file completely due to the limited network access and controlled environment. Another study (Bolton and Anderson-Cook, 2017) developed a random forest to classify malware families by applying blacklist, bigram instruction comparison, and call graph similarity comparison. The work (Rosenberg et al., 2017) argued that static analysis features extracted from reverse engineering require a large amount of pre-processing and hand-engineered domain-specific features to obtain relevant features. It applied a CNN model and the features extracted from dynamic analysis to classify malware authorship.

2.3 Convolutional neural network

A CNN (LeCun et al., 1999) is a fully connected multilayered artificial neural network, where each neuron is in a layer is connected to all neurons in the next layer. CNNs employed popularly for image recognition can recognise multiple objects without explicit segmentation of the objects from their surroundings. The work (Donahue et al., 2014) proposed a method of implementing deep convolutional activation features, and the proposed model can be considered as a deep architecture for transfer learning. The CNN learns a high-level hierarchy of the features during the training phase and the deeper the hidden layer is the higher the abstraction level of the features (Rosenberg et al., 2017).

3 Proposed detection method

The proposed detection method employs the following techniques to detect process-injection malware by using both static and dynamic analysis approaches.

1 The static analysis approach examines if executable files invoke the sensitive API function calls used by process-injection malware.

2 The Windows hooking mechanism monitors running processes and captures injected DLLs.

3 The CNN classification model examines if a captured DLL is malicious or not.

Figure 2 outlines the system architecture.

3.1 Static analysis

According to the literature review (Christodorescu et al., 2005; Idika and Mathur, 2007; Egele et al., 2012), this study concluded the sensitive API functions invoked by process-injection malware summarised in Table 2. For example, a malicious program invokes CreateRemoteThread(), SetWindowsHookEx(), and OpenProcess(), to inject a DLL into a victim program. The selected API functions cover the process injection attack techniques commonly used for DLL injection, remote execution, and registry modification. The proposed static analysis method disassembles executable files, extracts the invoked sensitive API calls, and applies an ML classification model to detect process-injection malware.

Figure 2 The proposed system architecture (see online version for colours)

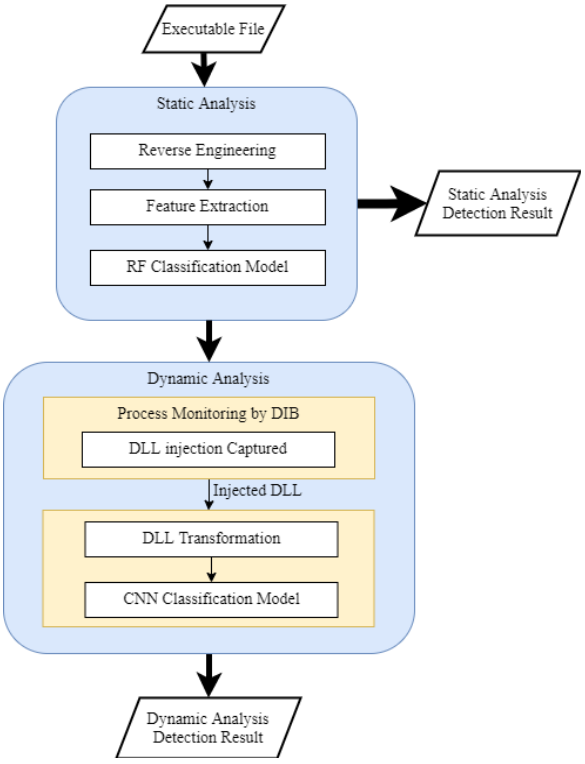


Table 2 Sensitive API functions used in process injection techniques

OpenProcess	VirtualAlloc
LoadLibrary	VirtualAllocEx
LoadLibraryE	CreateProcess
CreateRemoteThread	CreateRemoteThreadEx
WriteProcessMemory	SetWindowsHookEx
UnHookWindowsEx	CallNextHookEx
SetWindowEx	UnHookWinEvent
Shellexecute	WinExec
DLLFunctionCALL	RegCreateKey
RegSetValue	

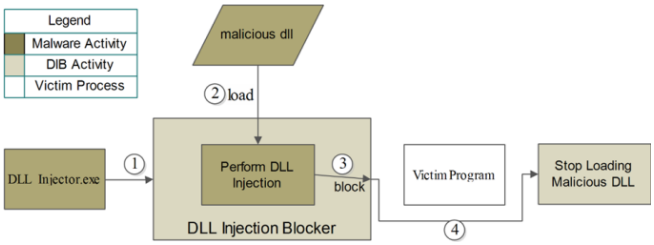
3.2 Dynamic analysis

By employing a process-injection technique, an attacker injects a malicious DLL into the memory space of a benign process, such as svchost, explorer, or other commonly used processes. As mentioned in the introduction, the intention of such injection depends on the injected code, not the technique, a benign program might adopt DLL injection to

invoke an updated library function, while malware injects a malicious DLL or shellcode for launching an attack. Both apply the same technique injecting a DLL, but each invokes a DLL with a different intention. Therefore, to improve detection efficiency, the proposed dynamic analysis method monitors process execution, captures injected DLLs, and examines if they are malicious or not.

The proposed dynamic analysis method consists of the following two steps. The first step, DLL injection blocker (DIB) (Long, 2017), applies the hooking technique to capture injected DLLs; the second step applies a CNN classification model to examine if the captured code is malicious or not. Figure 3 outlines the proposed DIB algorithm employing the Windows hooking technique to monitor process execution and suspends the process execution if it injects a DLL into the address space of another process.

Figure 3 The DIB flowchart (see online version for colours)



3.3 DLL classification model

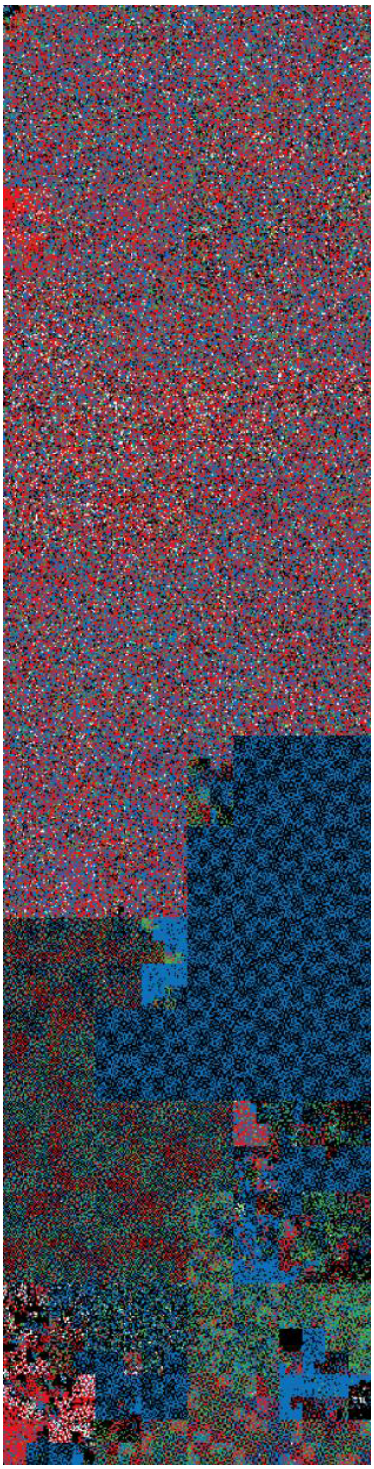
Based on the literature review, CNNs have been successfully applied to image recognition and classification. The past research (Nataraj et al., 2011) converted malware binary files into images and demonstrated that malware in the same family retains similar features. Therefore, this study employs CNN to classify malicious DLLs by transforming binary executable files into images, where executable files are transformed (visualised) into Hilbert space-filling curves by utilising the Scurve tool (Cortesi, 2015).

The Hilbert curve (1891) (or Hilbert space-filling curve) is constructed as a limit of piecewise linear curves. The length of the n^{th} curve is 2n – 1 / 2n. Therefore, the length grows exponentially with n, and each curve is contained in a square of area 1. It is useful for presenting a locality-preserved mapping between 1-dimensional and 2-dimensional space (Moon et al., 2001), which implies that two data points that are close to each other in 1 dimension space are also close to each other after the transformation.

Colours divide byte values into the following segments: low byte, ASCII code, and high byte, where 0X00 and 0XFF are special colours, tab (0X09), line feed (0X0A), and enter (0X0D) are considered as characters. Figure 4 illustrates the image of wininet.dll.

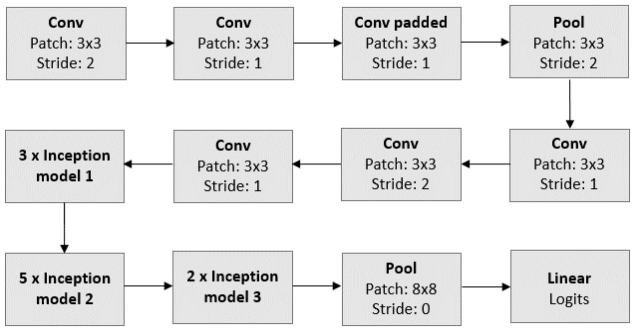
This study adopts an improved CNN model, the Inception V3 architecture (Szegedy et al., 2016) as it applies factorised convolutions and aggressive regularisation to improve computation efficiency and to reduce overfitting risk. It outperforms several modern models including VGGNet (Simonyan and Zisserman, 2015), GooLeNet (Szegedy et al., 2015), and PReLU-nets (He et al., 2015). The reduction of feature dimension can be viewed as a special case of factorising convolutions in a computationally efficient way. Based on the observation that the outputs of nearby activations are highly correlated, it develops smaller convolutions for factorisation. For example, a 5 × 5 convolution is replaced by two 3 × 3 convolutions; a 3 × 3 is replaced by one 3 × 1 and one 1 × 3 convolutions. Figure 5 illustrates the basic Inception V3 architecture.

Figure 4 Visualised wininet.dll (see online version for colours)



Transfer learning is efficient on model training with limited training data and allows you to retrain the final layer of an existing model. One of the most famous models used for transfer learning is Inception V3 (Milton-Barker, 2019). Based on the transfer learning property that the knowledge of solving one type of problem can be used to solve a similar problem (Donahue et al., 2014), this study visualises binary executables into Hilbert curve images and adopts the CNN classification model based on inception V3 architecture to identify malicious DLLs.

Figure 5 The basic architecture of inception V3



Source: Nguyen et al. (2018)

The training data set consists of benign and malicious DLLs, where the benign DLLs are obtained from a clean Windows 10 64-bit system and a Windows 10 64-bit system installed with anti-virus and commonly used software and the malicious DLLs are from the NCHC malware knowledge base. Based on our preliminary study, benign DLLs obtained from a clean Windows 10 64-bit system without installing any popular applications are not enough for training an effective malware classification model, as such a system might miss some common benign DLLs.

4 Performance evaluation

As the proposed solution combines static and dynamic analysis and ML classification models, therefore, this study designs four experiments to evaluate the efficiency of each adopted technique. Exp 1 is to validate the performance of the proposed static analysis method; Exp 2 is to examine how to train an effective CNN classification model; Exp 3 is to evaluate the efficiency of detecting unknown process injection malware; Exp 4 is to evaluate the detection efficiency of the proposed solution by comparing with VirusTotal.

The test data sets primarily consist of two parts: malicious samples and benign programs, where the 1,500 malware samples and 5,000 malicious DLLs are obtained from the NCHC malware knowledge base, Our preliminary study observes that the benign samples obtained from a clean Windows 10 64-bit system are limited and could not achieve good performance. Therefore, the benign programs and DLLs are extracted from a Windows 10 64-bit system installed with anti-virus and commonly used software. Depending on the purpose of evaluation, an experiment would apply a data set suitable for its need.

The performance measurements applied in this study are defined below. Detection rate or true positive rate (TPR) measures the proportion of actual malware that are correctly identified; true negative rate (TNR) measures the proportion of actual benign that are correctly identified; false negative (FNR) measures the proportion of actual malware that are misclassified; false positive (FPR) measures the proportion of actual benign that are misclassified. Accuracy (ACC) and TPR are used for evaluating the detection models as expressed below.

$$ACC = \frac{(TP)+(TN)}{(Tp)+(TN)+(Fp)+(FN)}$$

$$TPR = \frac{(TP)}{(TP)+(FN)}$$

4.1 Exp 1: evaluation of static analysis detection

The evaluation data set of Exp 1 consists of 1,500 malware samples and the same amount of benign programs. The experiment applies ten-fold evaluation with 6:4 ratio of training and testing, and the results are outlined in Table 3. Among different ML models, random forest yields the best detection performance.

4.2 Exp 2: training the detection model

Exp 2 evaluates the impact of the training data on model training. The detection performance of an ML-based model depends on the quantity and quality of the data set. Figure 6 shows the impact of a small data set, where the data set consists of 1,000 benign and 719 malicious. Figure 7 indicates the impact of imbalanced data with 2,038 benign and 719 malicious. The experimental results demonstrate that a small-sized data set or imbalanced data affects model training. Even though both trained models reach high precision, their validation data yields poor detection performance.

Table 3 The detection performance of static analysis

Classification model	*ACC*	*TPR*	*ROC area*
Naïve Bayes	82.49%	90.5%	0.943
J-48	93.46%	90.0%	0.964
Random forest	94.74%	91.0%	0.985
SVM	93.90%	91.6%	0.926

After several attempts of the adjustment, the evaluation data set of this experiment contains 5 K malicious samples and 5 K benign programs, and the experiment is performed by splitting it in 8:1:1 (training: testing: validation). Figure 8 plots the performance of the trained CNN detection model.

Figure 6 Precision drops in case of a small training dataset (see online version for colours)

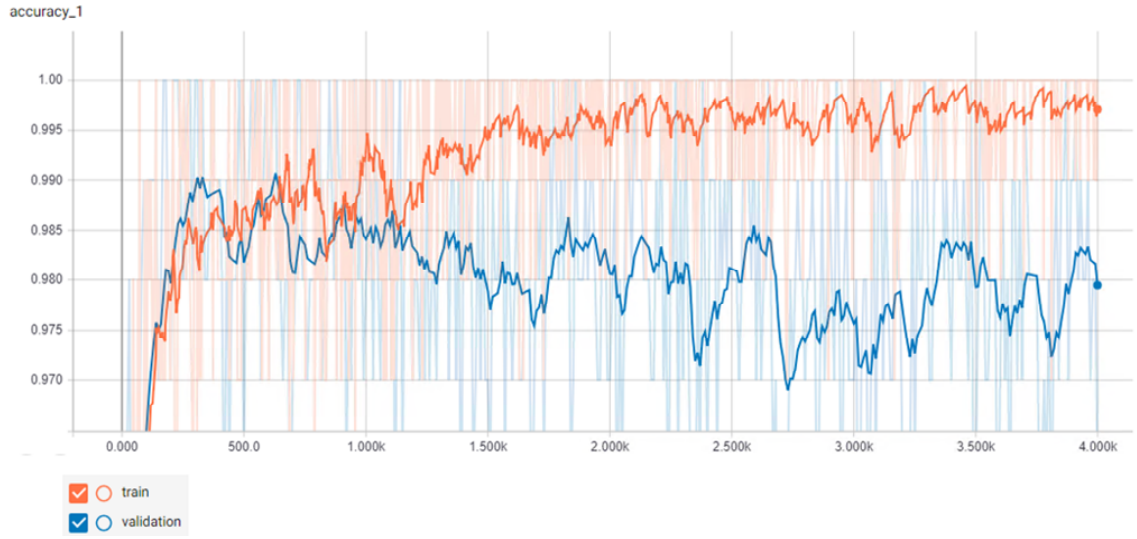


Figure 7 Precision drops in imbalanced data (see online version for colours)

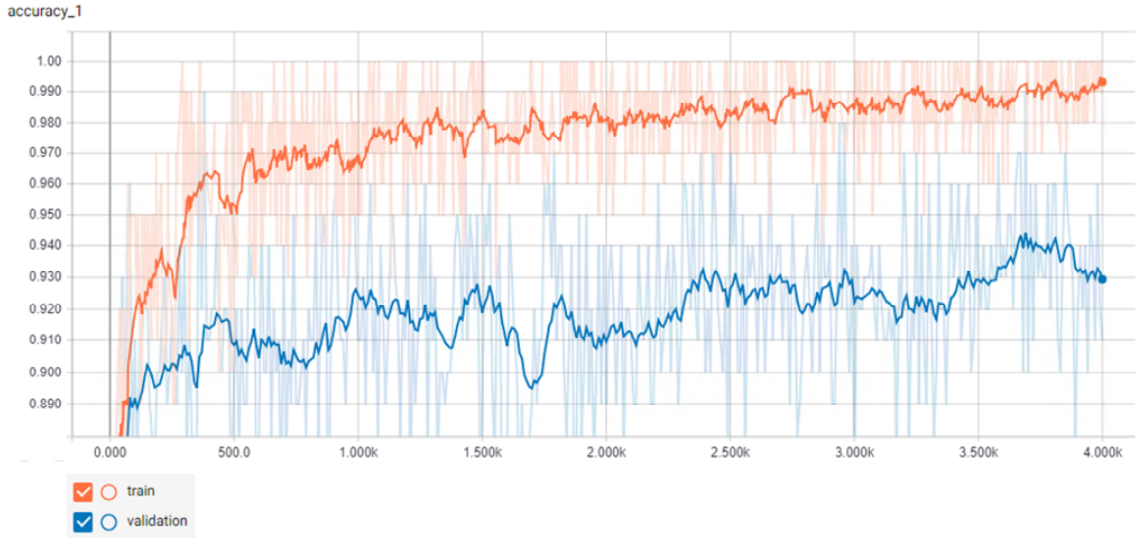


Figure 8 The performance of the trained CNN detection model (see online version for colours)

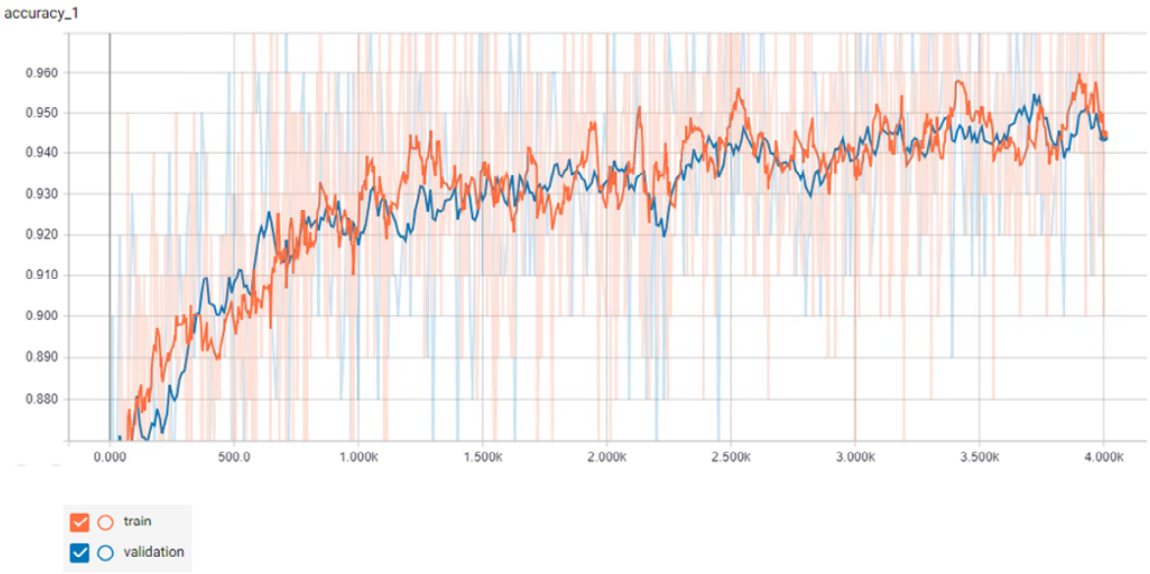


Figure 9 Malicious DLL is blocked (see online version for colours)

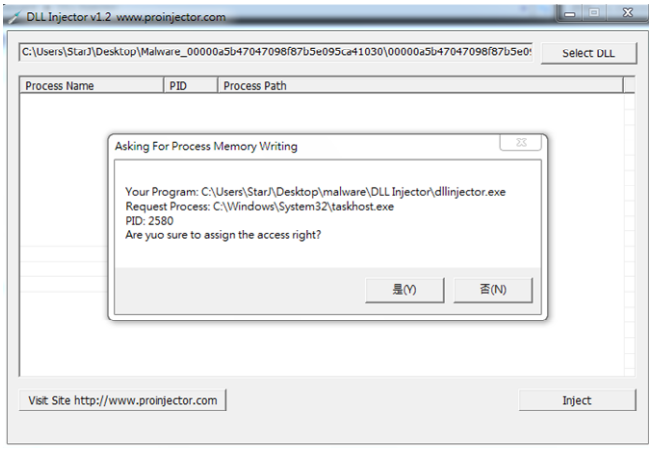
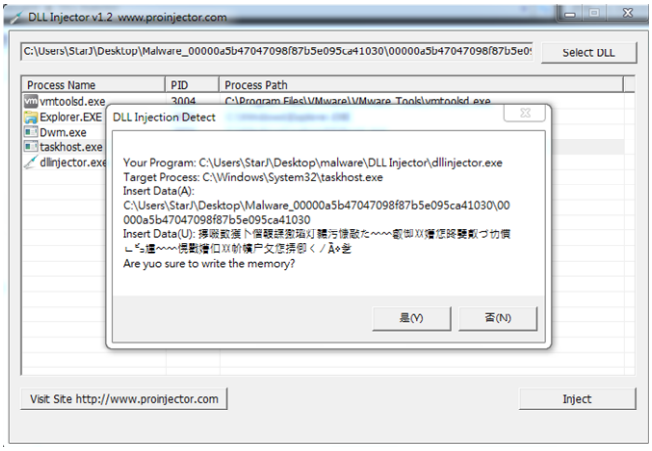


Figure 10 The attributes of the malicious DLL are recorded (see online version for colours)



4.3 Exp 3: detecting unknown malware

In order to evaluate if the proposed method could identify unknown process-injection malware, this experiment mimics a hacker utilising a DLL injector (Apponic, 2021) to create new malware injecting an unknown malicious DLL. The results show that the proposed system blocked it when it attempted to access the address space of another process and alerted the user if s/he wants to grant such access as shown in Figure 9. In addition, the system records the attributes of the blocked DLL as shown in Figure 10 and produces the associated activity records in the event log file as shown in Figure 11. In summary, this experiment demonstrates that the proposed system could successfully identify unknown malware and block its execution.

4.4 Exp 4: performance comparison

This experiment selects 52 malicious DLLs that have a low detection rate in VirusTotal in order to evaluate if the proposed system can identify them efficiently. The rationale of the malware selection is to validate if the proposed solution could outperform most of the commercial anti-virus software on detecting process-injection malware. Table 4 summarises the detection results of both systems. The results show that the proposed CNN-based detection system identifies 43 out of 52 with the recall of 82.7% and can detect malicious DLLs efficiently, while most commercial anti-virus software might not be able to.

Figure 11 The associated activities are recorded in the event log file (see online version for colours)

Table 4 Detection comparison with VirusTotal

Malware MD5	*Virus total*	*Our method*
0000ae2d955e1c0cd7374f931f5d88d0	2 / 56	Malicious
0001e298ef8bd956d03e688d101679e0	5 / 55	Malicious
00021589be9f0b8d7accb832446e5ae0	2 / 53	Malicious
000292d36ea2553188704fd524fe9e60	4 / 55	Malicious
00038620771ed92cd79d5269319b6a20	5 / 57	Malicious
0003e418265806944a84e7a9c54efa90	2 / 53	Benign
0004653825e92f2ab46bf023d7d80280	3 / 57	Malicious
010d661a35d42b5972bfd711d37cd0c0	12 / 53	Malicious
010e4e76d9674306a1cdf06de3484650	4 / 56	Malicious
010e6700f90558e84bcf1094908c2a30	9 / 54	Malicious
010e756d193f31537d9f974263170a90	5 / 56	Malicious
0110b9390d4bdad2186b6e76496c31b0	8 / 57	Benign
01136628fca38fe5ed0fb413bd410ea0	8 / 57	Benign
0113abf63083cbfe11282742884ab7e0	17 / 57	Malicious
0115d80c30f6ea75264cfb63c5f7f240	18 / 57	Malicious
0115fb83ac30a7dff67fd3feda006440	2 / 56	Malicious
01164f964969e1cd218b9e50cc78be70	16 / 56	Malicious
01177f6812e7b84f70ef8305fb9bbed0	11 / 53	Malicious
011aa50f69a124827376a30fb37dca50	10 / 54	Malicious
011f33a59eeaccd107e5dd679607e500	18 / 57	Malicious
0120d67c9e9e2e9d59a04a58778b06c0	13 / 56	Benign
01249c46fb111b390a48f75338a9c590	14 / 55	Malicious
01268a82e47909cc012343f29a4dcbf0	2 / 54	Malicious
013139d855f3a6ac60701f08a7e00370	9 / 56	Malicious
013190175ad11f71ffb49448104c6e30	8 / 55	Malicious
013342eadd62900cf9ba8d5cb46761a0	9 / 57	Benign
0139b85d75a7cb38bd833c073630ff20	19 / 56	Malicious
013a0da8ad277266f1ebe1595a2aea90	10 / 64	Malicious
013ce98a968c90b5a0056140ffa8f030	2 / 55	Benign
0141b4983a69de9dbfa6b326c8464720	2 / 55	Malicious
0141ec6335cf120275d9073c08c40ee0	3 / 55	Malicious
014b77afb6c0f89cdb51e39ec85acda0	5 / 57	Benign
014cb9f67365cba19087129152cd1530	4 / 56	Malicious

Table 4 Detection comparison with VirusTotal (continued)

Malware MD5	*Virus total*	*Our method*
014ed968d05aecef92a0c7e888de0a80	2 / 56	Benign
015006ffe06f1fa250f1bc80438f99d0	9 / 57	Malicious
0151eba491d1049e40a933ca3a988210	4 / 53	Malicious
01525f3479ddad7de03e9ffc2b584720	11 / 54	Malicious
0153fa7457e860acb60261b3a0b05140	13 / 55	Malicious
01585b80d4bbb920f9567c065e62ac70	6 / 56	Malicious
015a30f1820ed04d7dd2cb544a7020e0	4 / 56	Malicious
015b28d5279db3bf15ae578b375a13a0	3 / 56	Malicious
015f2e599caea9c733a407d44d4ad430	4 / 53	Benign
0161ff82af4921885fa3badab3877bf0	4 / 52	Malicious
01639510ad6999aa9284a3352eede8d0	4 / 55	Malicious
016504076f6f5ca83dfb49a508f38870	5 / 56	Malicious
01655948f1028f906004085e589138e0	4 / 54	Malicious
0166d6e6eedb409a6c059ce5c2c44630	10 / 55	Malicious
016903a9bac7a6dff37528f82bcd2a40	9 / 57	Malicious
016c89ad2736e4df10610928489aa6d0	8 / 55	Malicious
0172c674da767ef1a4acbf5ee5cf0570	2 / 53	Malicious
0176d6fbd3f0ab30a5f8e81acaa89b90	3 / 55	Malicious
01771b93232c44f5bee57a69f6d17240	6 / 55	Malicious

5 Conclusions

Several process-injection techniques to create such malware. Some perform the injection behavior during execution, which can be captured only at run time; some can be identified through static analysis. To improve detection performance and time efficiency, this study proposes a process-injection malware detection approach that combines dynamic and static analysis as well as ML techniques.

This study summarises the sensitive API functions used by process-injection malware. The experimental results prove that the static analysis approach by using the sensitive API functions is effective in detecting process-injection malware and random forest yields the best detection performance. The proposed static analysis approach can be used for large scan, such as disk scan to inspect if process-injection malware exists in the disk.

Attackers can utilise injector tools to create new process-injection malware to evade detection. Furthermore, the detection rate of malicious DLLs is not high in most commercial AV software. By utilising transfer learning, the proposed system transforms binary executable files into images and applies a CNN classification model to identify malicious DLLs. The experimental results demonstrate that the proposed detection method could capture new process-injection malware and outperforms most AV software.

The proposed approach also can be considered as two-layer protection, where the first layer applies static analysis to perform disk scan and the second layer applies dynamic analysis to monitor and capture if any malware injects DLL during execution.

References

Angelystor (2020) *Process Injection Techniques Used by Malware* [online] https://medium.com/csg-govtech/process-injection-techniques-used-by-malware-1a34c078612c (accessed 1 June 2021).

Antoniewicz, B. (2013) *Windows DLL Injection Basics*, Open Security Research [online] http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html (accessed 15 February 2021).

Apponic (2021) *Injector* [online] http://injector.apponic.com/ (accessed 1 June 2021).

Bolton, A.D. and Anderson-Cook, C.M. (2017) 'APT malware static trace analysis through bigrams and graph edit distance', *Statistical Analysis and Data Mining*, Vol. 10, No. 3, pp.182–193.

Cesare, S., Xiang, Y. and Zhou, W. (2013) Malwise – an effective and efficient classification system for packed and polymorphic malware', *IEEE Transactions on Computers*, Vol. 62, No. 6, pp.1193–1206.

Chang, T.C. (2016) *Detecting Malware with DLL Injection and PE Infection*, MS thesis, MIS of NSYSU, Kaohsiung.

Chang, Y.H. and Singh, S. (2016) *APT Group Sends Spear Phishing Emails to Indian Government Officials* [online] https://www.fireeye.com/blog/threat-research/2016/06/apt_group_sends_spea.html (accessed 15 February 2021).

Choi, Y.H., Han, B.J., Bae, B.C., Oh, H.G. and Sohn, K.W. (2012) 'Toward extracting malware features for classification using static and dynamic analysis', Paper presented at the *2012 8th International Conference on Computing and Networking Technology (ICCIS)*.

Christodorescu, M., Jha, S., Seshia, S.A., Song, D. and Bryant, R.E. (2005) 'Semantics-aware malware detection', Paper presented at the *2005 IEEE Symposium on Security and Privacy (S&P'05)*.

Cortesi, A. (2015) *Scurve* [online] https://github.com/cortesi/scurve (accessed 12 October 2020).

dirty-needle (2017) [online] https://github.com/lillypad/dirty-needle (accessed 12 October 2020).

DLL-Inj3cti0n (2015) [online] https://github.com/nyx0/DLL-Inj3cti0n (accessed 12 October 2020).

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E. and Darrell, T. (2014) 'Decaf: a deep convolutional activation feature for generic visual recognition', Paper presented at the 31st *International Conference on Machine Learning*, pp.647–655.

Egele, M., Scholte, T., Kirda, E. and Kruegel, C. (2012) 'A survey on automated dynamic malware-analysis techniques and tools', *ACM Computing Surveys*, Vol. 44, No. 2, pp.1–42.

Gorelik, M. (2017) *Fileless Malware Attack Trend Exposed* [online] https://www.morphisec.com/hubfs/wp-content/uploads/2017/11/Fileless-Malware_Attack-Trend-Exposed.pdf (accessed 15 February 2021).

Granneman, J. (2013) 'Antivirus evasion techniques show ease in avoiding antivirus detection' [online] https://searchsecurity.techtarget.com/feature/Antivirus-evasion-techniques-show-ease-in-avoiding-antivirus-detection (accessed 15 February 2021).

Hăjmăşan, G., Mondoc, A. and Creţ, O. (2017) 'Dynamic behavior evaluation for malware detection', Paper presented at the *2017 5th International Symposium on Digital Forensic and Security (ISDFS)*.

He, K., Zhang, X., Ren, S. and Sun, J. (2015) 'Delving deep into rectifiers: Surpassing human-level performance on imagenet classification', Paper presented at the *2015 IEEE International Conference on Computer Vision (ICCV)*.

Hilbert, D. (1891) *Ueber die stetige Abbildung einer Linie auf ein*, Mathematische Annalen.

Hosseini, A. (2017) 'Ten process injection techniques: a technical survey of common and trending process injection techniques' [online] https://www.elastic.co/blog/ten-process-injection-techniques-technical-survey-common-and- trending-process (accessed 15 February 2021).

Idika, N. and Mathur, A. (2007) *A Survey of Malware Detection Techniques*, Purdue University, p.48.

Ijaz, M., Durad, M. H. & Ismail, M. (2019). Static and Dynamic Malware Analysis Using Machine Learning. Paper presented at the 2019 16th

Injection (2019) [online] https://github.com/theevilbit/injection (accessed 15 February 2021).

InjectProc (2019) [online] https://github.com/secrary/InjectProc (accessed 15 February 2021).

Kaspersky (2021) *Dealing with Svchost.exe Virus' Sneak Attack, Kaspersky* [online] https://www.kaspersky.co.uk/resource-center/threats/dealing-with-svchost-exe-virus-sneak-attack (accessed 1 June 2021).

Kindlund, D. (2013) *POISON IVY: Assessing Damage and Extracting Intelligence* [online] https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-poison-ivy.pdf (accessed 1 June 2021).

Kozachok, A.V. and Kozachok, V.I. (2018) 'Construction and evaluation of the new heuristic malware detection mechanism based on executable files static analysis', *Journal of Computer Virology and Hacking Techniques*, Vol. 14, pp.225–231.

LeCun, Y., Haffner, P., Bottou, L. and Bengio, Y. (1999) 'Object recognition with gradient based learning', *Springer LNCS*, Vol. 1681, pp.319–345.

Long, J.S. (2017) *DLL Injection Detection in Runtime*, MS thesis, MIS of NSYSU, Malware Knowledge Base, Kaohsiung.

Milton-Barker, A. (2019) *Inception V3 Deep Convolutional Architecture for Classifying Acute Myeloid/Lymphoblastic Leukemia* [online] https://software.intel.com/content/www/us/en/develop/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute- myeloidlymphoblastic.html (accessed 1 June 2021).

Minerva (2021) *Memory Injection Prevention, Prevent Fileless and Other in Memory Attacks* [online] https://minerva-labs.com/memory-injection (accessed 1 June 2021).

Mitre (2021) *Process Injection*, MITRE ATT&CK [online] https://attack.mitre.org/techniques/T1055/ (accessed 1 June 2021).

Moon, B., Jagadish, H.V., Faloutsos, C. and Saltz, J.H. (2001) 'Analysis of the clustering properties of the Hilbert space-filling curve', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13, No. 1, pp.124–141.

Nataraj, L., Karthikeyan, S., Jacob, G. and Manjunath, B.S. (2011) 'Malware images: visualization and automatic classification', Paper presented at the *8th International Symposium on Visualization for Cyber Security*.

Ngo, Q.D., Nguyen, H.T., Le, V.H. and Nguyen, D.H. (2020) 'A survey of IoT malware and detection methods based on static features', *ICT Express*, Vol. 6, No. 4, pp.280–286.

Nguyen, L.D., Lin, D., Lin, Z. and Cao, J. (2018) 'Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation', Paper presented at the *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*.

Polino, M., Scorti, A., Maggi, F. and Zanero, S. (2015) 'Jackdaw: towards automatic reverse engineering of large datasets of binaries', Paper presented at the *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2015*, Springer, pp.121–143.

ProcessInjection (2020) [online] https://github.com/3xpl01tc0d3r/ProcessInjection (accessed 15 February 2021).

ProcessInjectionTool (2019) [online] https://github.com/Lexsek/ProcessInjectionTool (accessed 15 February 2021).

Rezaei, S., Afraz, A., Rezaei, F. and Shamani, M.R. (2016) 'Malware detection using opcodes statistical features', Paper presented at the *2016 8th International Symposium on Telecommunications (IST)*, IEEE.

Rosenberg, I., Sicard, G. and David, E. (2017) 'DeepAPT: nation-state APT attribution using end-to-end deep neural networks', Paper presented at the *International Conference on Artificial Neural Networks (ICANN), Springer LNCS*, Vol. 10614, pp.91–99.

Rosenquist, M. (2015) *Malware Trend Continues Relentless Climb* [online] https://www.linkedin.com/pulse/malware-trend-continues-its-relentless-climb-matthew-rosenquist (accessed 15 February 2021).

Sihwail, R., Omar, K. and Ariffin, K.A.Z. (2018) 'A survey on malware analysis techniques: static, dynamic, hybrid and memory analysis', *International Journal on Advanced Science Engineering and Information Technology*, Vol. 8, Nos. 2–4, p.1662.

Sikorski, M. and Honig, A. (2012) *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*, No Starch Press, US.

Simonyan, K. and Zisserman, A. (2015) 'Very deep convolutional networks for large-scale image recognition', Paper presented at the *3rd International Conference on Learning Representations*.

Srinivas (2021) *Antivirus Evasion Tools* [online] https://resources.infosecinstitute.com/antivirus-evasion-tools/#gref (accessed 15 February 2021).

Statista (2021) *Desktop PC Operating System Market Share Worldwide*, from January 2013 to July 2020 [online] https://www.statista.com/statistics/218089/global-market-share-of-windows-7/#:~:text=Desktop%20PC%20OS% (accessed 1 June 2021).

Sun, Y., Jee, K., Sivakorn, S., Li, Z., Lumezanu, C., Korts-Parn, L., Wu, Z., Rhee, J., Kim, C.H., Chiang, M. and Mittal, P. (2020) 'Detecting malware injection with program-DNS behavior', Paper presented at the *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D, Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) 'Going deeper with convolutions', Paper presented at the *IEEE Conference on Computer Vision and Pattern Recognition*, pp.1–9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2016) 'Rethinking the inception architecture for computer vision', Paper presented at the *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

TaNet (2021) *TANet Computer Emergency Response Team* [online] https://cert.tanet.edu.tw/prog/Document.php (accessed 1 June 2021).

WinPIT (2018) [online] https://github.com/m4mm0n/WinPIT (accessed 15 February 2021).

Ye, Y., Li, T., Adjeroh, D. and Iyengar, S.S. (2017) 'A survey on malware detection using data mining techniques', *ACM Computing Surveys (CSUR)*, Vol. 50, No. 3, pp.1–40.