Resolving Malware-Loading Problem by Leveraging Shellcode Detection Heuristics

Michael Cheng Yi Cho
National Chiao Tung University
michcho.cs98g@g2.nctu.edu.tw

Zhi-Kai Zhang
National Chiao Tung University
skyzhang.cs99g@g2.nctu.edu.tw

Shiuhpyng Shieh
National Chiao Tung University
ssp@cs.nctu.edu.tw

Abstract - Dynamic malware analysis platform is developed to extract malicious behavior from malware without the knowledge of its internal structure. Based on the analysis results, security experts can derive a signature for future detection. However, a malware program may not be invoked unless certain conditions are met. Therefore, the basic requirement for dynamic malware analysis is able to determine the conditions to activate a malware program. For execution, malware is just like any other software that may require additional supplementary information, such as parameters, arguments, and entry point. Therefore, loading malware with appropriate conditions to activate its malicious behavior can be a challenge. This paper resolves the malware-loading problem by leveraging shellcode detection heuristics based on remote exploit traffic. A formal model is proposed to model the malware loading problem. To cope with the problem, a new scheme is proposed, implemented and evaluated using malware samples collected from the wild. As the evaluation showed, the scheme is highly effective. 15% of the malware samples concealed their activities initially when loaded. After the proposed scheme is applied, 97% of the malware samples demonstrated activities.

Keywords - DLL malware; dynamic malware analysis; honeypot; shellcode detection.

I. INTRODUCTION

The invention of the Internet allows users to exchange information in a more convenient way. Along with the convenience, it also provides a medium for malware distribution. One of the malware distribution methods is exploiting vulnerable network service applications. This type of exploitation is known as *remote exploit*. To understand remote exploit, research [1–4] has described common scenarios for attackers to exploit network security vulnerabilities. The common scenarios can be divided into three phases: *pre-attack*, *attack*, and *post-attack* phases. The pre-attack phase gathers information of a targeted victim for the preparation of exploitation. The gathered information typically includes the type and version of a network service. The exploitation occurs in the attack phase where an attacker uses the gathered information to craft remote exploits to attack the vulnerable network service. The post-attack phase is to achieve the objectives of an attacker, such as data exfiltration, internal network system infection, and connect-back shell.

The attack phase can be divided into two procedures according to recent research [1][5][6], that is, vulnerability exploitation and egg-download. The vulnerability exploitation sets up malware download (egg-download) from a remote host and executes the downloaded malware to achieve the objectives of an attacker. This exploitation approach is designed to circumvent buffer limitation to achieve sophisticated post-attack objectives while maintaining flexibility of coupling exploit code with malware. In this particular scenario, two software programs are used to carry out the exploitation. One is the program to stage malware download and malware execution, conventionally called shellcode. The other is the program that carries out the objective of post-attack phase, namely malware.

Leverage the aforementioned attack pattern, Baecher et al. [5] presented a passive honeypot that simulates security vulnerabilities to capture wild malware. The captured malware is sent to a third-party dynamic malware analysis platform (namely sandboxes [7–9]) for malicious behavior analysis. Unfortunately, analyzing the captured malware can be a tedious task due to the assumption of the dynamic malware analysis. The assumption of dynamic malware analysis is based on the execution ability of malware. In other words, it is useful only if the malware can be executed on the dynamic malware analysis platform. To narrow down the research scope, we will look at malware on Windows platform specifically. Windows malware samples are generally delivered in 3 different executable formats [10], namely, executables, drivers, and DLLs. Although the loading methods differ amongst the file types, additional supplementary information is often needed for execution, such as input arguments, system parameters and entry points. Without the required supplementary information, the behavior of malware is concealed, and cannot be analyzed directly on the dynamic malware analysis platform. For simplicity, we will name the problem as *malware-loading problem* throughout this paper.

Malware execution can be context sensitive where malicious behaviors can only be observed when certain conditions are met. Research [11–13] addressed this problem and suggested that different behaviors can be observed when multiple execution traces are available. The approach is based on concolic testing [14][15] where condition branch of a piece of malware is labeled and conditions are computed to extend code coverage of malware execution. Although its targeted problem is related to execution-condition finding within a program, the malware-loading problem incurred in the shellcode is not addressed. We leverage shellcode detection [16–18] which detects the existence of shellcode in a file. Our proposed scheme locates the conditions needed to execute a malware program. It relies on shellcode detection to resolve the URI from the detected shellcode to locate and download the malware for further analysis. Our approach ensures that the malware is executed, rather than exploring code coverage, for triggering malicious behaviors.

The contribution of this research is two-fold. First, a formal model is derived to model the malware loading problem, and a solution to the problem is given. Second, an automatic execution framework is proposed that prepares the captured malware for dynamic malware analysis. Since the framework is automatic, it will save time and human resources on preparing malware for dynamic malware analysis. Real malware samples from the wild are used to evaluate the propose solution. The evaluation demonstrates that the propose solution is effective.

The reminder of this paper is organized as follows. In section 2, related work is discussed. Section 3 covers the background knowledge related to shellcode detection and dynamic malware analysis platform. Formal model of the research problem and its solution are discussed in section 4. Section 5 contains system design and implementation while section 6 presents the result of evaluation using wild malware samples. In section 7, limitation of the propose system is discussed and section 8 summarizes the results of this paper.

II. RELATED WORK

Although the malware-loading problem remains unsolved, the following related work may help understand the problem, namely shellcode detection, dynamic malware analysis, and code coverage for malware discovery.

Shellcode detection research focuses on detecting shellcode in an arbitrary file. The goal of this research is to discover shellcode traits in any given arbitrary file. Since the majority of shellcode is encrypted to evade static signature detection, most of shellcode research relies on dynamic execution to locate traits of shellcode. Research [16–18] relies on x86 emulator [19] to locate the traits of shellcode. Traits are various implementations of GetPC code [20] and suspicious techniques to memory location of APIs. These are crucial information for malicious shellcode programmers to perform useful attacks. Although these coding tricks are applicable to any given program/shellcode, an authorized program author does not have to take the longer route to obtain this program information.

Dynamic malware analysis is used to extract malware behavior by executing the targeted malware. Researchers [7–9] dedicated their research effort to the correctness of extracting execution behaviors. These research efforts are vital for anti-virus company to develop antidote to remove malware from a victim as well as developing detection signatures to avoid infection. The problems of malware execution and trigger-based malware are still not fully addressed in this research domain.

Exploring malware code coverage [11–13] mixes symbolic and concrete execution to determine the execution conditions to trigger malicious behavior when malware executes. The concept is based on concolic testing [14][15] where the code coverage of analysis is maximized for software reliability. However, without the knowledge of trigger type, the process of finding the solution can be time consuming and resource extensive. Our proposed solution aims to quickly determine execution conditions for activating malware. It is to complement, not to replace trigger-based malware condition identification research.

III. BACKGROUND KNOWLEDGE

Malware is built to install itself into a victim and perform malicious acts such as infiltration, building botnets, and taking control of the victim. To infect a victim with a piece of malware, an attacker will need to trick the victim to install the desirable malware. The methods can be divided into active and passive from attacker's prospective. The active method leverages security vulnerabilities to install desirable malware. On the other hand, the passive method requires media to trick users to download and install malware. In this scenario, the attacker will need to wait until a victim responds. In this paper, we limit our scope to an active method called arbitrary code execution which leverages memory corruption vulnerabilities remotely.

As mentioned in section I, a remote exploit which preforms arbitrary code execution on memory corruption vulnerability often involves two pieces of software: shellcode and malware. Shellcode is a code segment to stage malware installation and malware execution while the malware carries out post-attack objectives. Shellcode is usually compact in size to fit into a network packet. It uses limited application programming interface (API) functions to be highly portable. One particular common choice of API functions that appear in wild shellcode samples is LoadLibrary and GetProcAddress from Kernel32.dll [21–23]. As the names suggested, these APIs are used to load library dynamically and locate the API function of the loaded libraries while Kernel32.dll is loaded for all Windows applications. Since shellcode exercises behaviors to search for these APIs, research [16–18] used the behaviors as signatures to detect shellcode existence in a file or a network packet. Other detection methods also exist in the described literatures which is based on the idea that shellcode attempts to discover the current memory location it resides. This information is particularly needed if the shellcode adopts code obfuscation methodology to determine the memory location the obfuscated code resides. The GetPC code [20] will not be encrypted given the need of direct execution. Hence, it is an effective approach to locate the shellcode.

Dynamic malware analysis [7–9] adopts the strategy of recording the execution behavior in a controlled environment. The method is useful in capturing malware behaviors. However, the activation of malware is the basic requirement to use dynamic malware analysis method. A piece of software is defined as malware if the software performs malicious functions. Just like any software, malware needs additional supplementary information to run, such as parameters, arguments, and entry point. Without the information, malware cannot be executed on a dynamic malware analysis platform. Therefore, we propose an activation scheme which leverages shellcode detection results to provide execution information needed for the activation of targeted malware.

IV. THE PROBLEM

As aforementioned, additional supplement information may be required in order to activate malware. Without these required information, malware cannot be analyzed on the dynamic malware analysis platform. Therefore, we leverage the knowledge of shellcode detection research domain to extract the required information. The extracted information can be used as an aid for dynamic malware analysis to extract malicious behaviors of a studied malware. The problem is formally defined as follows.

Computer program can be modeled as *Mealy* machine where a set of inputs and a set of states generate a set of outputs. Since the problem of interest regards to computer programs (malware), it is appropriate to model the problem using *Mealy* machine.

Definition 1: a FSM (Finite State Machine) is a quintuple

$$M = (I, O, S, \delta, \lambda)$$

where $I, O,$ and S are finite and nonempty sets of input symbols, output symbols, and states, respectively.

$s_0 \in S$ giving s_0 is the initial state
$\delta : I \text{ X } S \rightarrow S$ is the state transition function
$\lambda : I \text{ X } S \rightarrow \text{O}$ is the output function

Majuscule letters representing sets and minuscule letters representing elements.

We use *Definition 1* to define the property of dynamic malware analysis. Dynamic malware analysis adopts the concept of black-box testing, where the analysis only focus on the generated outputs of the malware and the internal structure of malware is not studied. Thus, we define the inputs and the outputs of dynamic malware analysis for the research problem in *Definition 2*.

Definition 2: input and output of dynamic malware analysis
Suppose the target of interest for dynamic malware analysis is M_{mal}

$$M_{mal} = (I_m, O_m, S_m, \delta_m, \lambda_m)$$

Input:

$s_{(m,0)}$ as the initial state of M_{mal}
$i_{(m,0)}$ as the input symbol of $s_{(m,0)}$

Output:

O_m as any alternations of memory region

According to *Definition 2*, the necessary conditions to apply dynamic malware analysis are the starting state of the targeted malware and the corresponding input for the starting state. The conditions refer to parameters/arguments as the input symbols and the entry point as the starting state of the malware under investigation. The outputs refer to execution behavior such as add/remove/modification of main memory, files, etc. Hence, knowing the starting state and input symbols of starting state is crucial to apply dynamic malware dynamic analysis.

Before we model a solution to the problem, we need to define the assumptions of remote exploits that leverage buffer over flow vulnerability. According to research [1–4], there are network exploits that downloads malware to infect victims. Within this exploitation, the following assumptions are made,

- A piece of shellcode is included as exploitation input
- The shellcode is responsible for malware download and downloaded malware execution

A program vulnerable to memory corruption can be modeled as follows according to *Definition 1*. Note that the vulnerabilities of a program often appear with careless implementation of the original FSM. Given careless implementation of FSM, we make use of *prime* to represent the elements that are different to the original FSM.

Let M' be a program that is prone to memory corruption such that

$$M' = (I', O', S', \delta', \lambda')$$

$i' \in I'$ where $i'$is an input that causes memory corruption
$s' \in S'$ where s$'$ is a state that contains memory corruption vulnerability
$o' \in O'$ where $o'$is the output generated by the state prone to memory corruption
$\delta' : i' \text{ X } s' \rightarrow s_{error}$ where s_{error} is the state after memory corruption
$\lambda : i' \text{ X } s' \rightarrow o'$ where o' is generated after M' is exploited

Majuscule letters representing sets and minuscule letters representing elements.

According to the defined assumption, there are three FSM involved in a scenario of memory corruption exploitation. They are:

$M' = (I', O', S', \lambda', \delta')$ as a program that is prone to memory corruption due to erroneous implementation

$M_{shell} = (I_s, O_s, S_s, \lambda_s, \delta_s)$ as a piece of shellcode that carried out the actions of malware download and downloaded malware execution

$M_{mal} = (I_m, O_m, S_m, \lambda_m, \delta_m)$ as a piece of malware that is downloaded and executed by M_{shell}; it is also the target for dynamic malware analysis

The relations of the three FSMs are described as follows:

For $M'$to execute M_{shell} given
$\delta' : s' \text{X } i' \rightarrow s_{error}$ exists in M'
i' contains M_{shell}
$s_{error} \in S_s$ where s_{error} is identical to $s_{(s,0)}$

M_{shell} can be executed independently if
$\delta' : s' \text{ X } i' \rightarrow s_{error}$ of M' is identified
$\emptyset \subseteq I_s$ and I_s is independent of O' of M'
$i_{(s,0)} \in \emptyset$ where the initial input symbol of M_{shell} is null

Since M_{shell} is responsible for downloading and executing M_{mal}, M_{shell} individual execution will cause M_{mal} execution if dependency does not exists between M' and M_{shell}. A typical program dependency is the context of a given program which is the context of the current memory snapshots. Therefore, if M_{shell} and M' are independent; direct execution of M_{shell} will cause M_{mal} download and M_{mal} execution.

V. PROPOSED SCHEME

The goal is to extract shellcode from remote exploit network traffic to execute the downloaded malware. The execution behavior can be reconstructed by taking shellcode directly from the remote exploit packet, which is recorded network stream during remote honeypot exploitation. The recorded network streams are passed into shellcode detection algorithm to identify the offset of the shellcode. Upon shellcode identification, the shellcode is extracted from the recorded network stream and packed as an executable program. The generated executable program is ready to be analyzed in a dynamic malware analysis platform. To ensure Sandbox evaluation process operates malware download action accordingly, a gateway router is set up redirect the malware download network traffic to a customized malware host.

Figure 1 depicts system relations between the propose scheme, honeypot, and dynamic malware analysis system. Honeypot captures wild malware while the dynamic malware analysis platform analyzes the captured wild malware to develop signature for system protection. As aforementioned, a portion of the captured wild malware cannot be directly executed on the dynamic malware analysis platform where the captured wild malware required additional supplementary information for execution. Therefore, we propose a new scheme which interacts between honeypot and dynamic malware analysis platform to assist the malware analysis process.

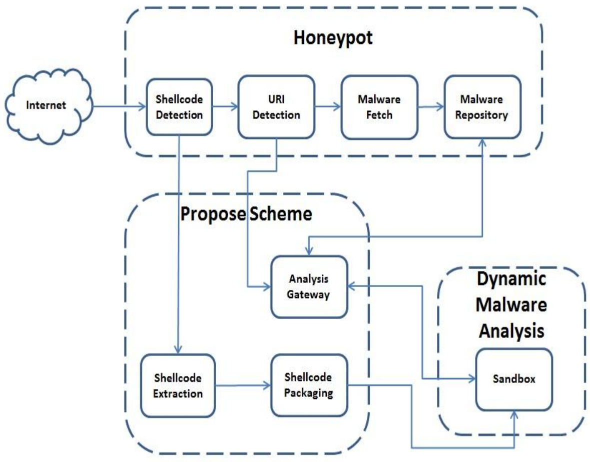

Fig. 1. System architecture of the propose scheme

There are three major components in the propose scheme; shellcode extraction, shellcode packaging, and analysis gateway. The shellcode extraction component leverages shellcode detection algorithm to locate the shellcode offset value of the incoming packet. The offset value is used to extract the shellcode as the input for shellcode packaging component. The shellcode packaging component will package the shellcode as an executable file that can be directly executed on the targeted dynamic malware analysis platform. The last component, analysis gateway, will act as the network gateway for shellcode execution when shellcode is analyzed on the dynamic malware analysis platform. In order to serve the corresponding malware when the executable shellcode made the request, the analysis gateway component must preserve the information of the Uniform Resource Identifier (URI) detection component on the honeypot. The preserved information is later used to serve the corresponding malware when the request is made by the captured shellcode.

As the result of execution of packaged shellcode on the dynamic malware analysis platform, it will trigger the behavior as the exploitation designed originally if the shellcode has no dependency on the exploited network service. The expected behavior will include download of malware and execution of the downloaded malware. Both behaviors will be recorded and reports will be generated accordingly for information security experts to develop the signature for the analyzed malware.

VI. EVALUATION

The evaluation samples are collected by deploying Dionaea [24], a honeypot, in the National Chiao Tung University (NCTU) campus network. The honeypot uses public Internet Protocol (IP) address and collected wild malware for over one year. The total collection of the wild malware samples is 343 distinctively, and over 15,000 exploits were recorded during the operation period. The wild malware samples are downloaded from 807 distinctive URI from 755 unique IP addresses. The wild malware samples are mainly delivered by using Hypertext Transfer protocol (HTTP) and Service Message Block (SMB) protocol, and the statistics are 658 and 149 respectively.

Figure 2 contains the file type distribution of the collected wild malware samples. There are 275 executable malware samples and 68 data files in total. Within the wild executable malware samples, 247 executable files are delivered by HTTP individually, and the other 28 executable files are delivered with 68 data/raw files which are carried by SMB. After investigating the exploitation of the malware samples, the exploitation can be divided into two categories based on the results of exploitation. Category one achieved arbitrary code execution [24][25], while the other category achieved remote code execution [26][27]. The difference of the two categories is divided by the existence of shellcode when vulnerability is exploited where the former results in arbitrary code/shellcode execution and the latter called the library API to achieve malicious acts. The malware samples delivered by HTTP belong to category one and the malware samples delivered by SMB belong to the other category. Since the propose scheme is shellcode-based, this evaluation will only focus on the 247 executable malware samples. Figure 2 also demonstrates the ratio of successful malware execution without prior knowledge of the execution condition. About 15% of the wild DLL malware samples and 38% of the wild executable malware samples require additional information for malware execution. Again, only the DLL malware samples are eligible for this evaluation; therefore, 15% of the evaluation malware samples require additional information for dynamic malware analysis.

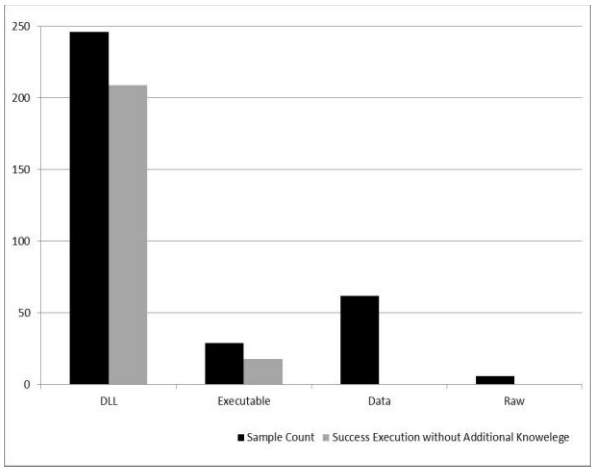

Fig. 2. Sample malware file type distribution

Using the implementation of the proposed scheme, the shellcode of the collected remote exploits is extracted and wrapped as Windows binary executables for dynamic malware analysis. The dynamic malware analysis platform for the evaluation is called Malware Behavior Analyser (MBA) [28] [29] which is an enhancement of QEMU [30] platform. Like any other dynamic malware analysis platforms, MBA is designed to execute malware samples in a virtual environment to record the execution behaviors including alternation of the file system and registry, and packet recording throughout the execution. Each of the evaluated shellcode samples is given a window period of 2 minutes to carry out the execution. After execution of the given window period, a report is generated from the MBA. The generated report is then summarized to evaluate the result of execution.

Since that are over 14,000 samples available for evaluation, an experiment environment is set up to carry out the experiment automatically. The experiment environment is based on distributed system concept where a master machine is set up to distribute evaluation samples to 13 individual slave machines with MBA pre-installed for evaluation. Once an evaluation task is completed on a slave machine, the report is sent back to the master machine for further report summarization. During the experiment, the evaluation sample will try to acquire the executable malware from a remote host as shellcode originally intended. The malware acquiring network traffic will be redirected to a customized malware repository host by a preconfigured router. The customized malware repository host is designed to read the HTTP request and respond accordingly to a preconfigured database.

As a result of the evaluation, 97% of the evaluated samples significantly displayed the signs of malware execution. After investigating the failure cases, the failed shellcode samples can be further grouped into three categories. In category one, the evaluated shellcode does not demonstrate the behavior of malware acquisition. This problem is due to delivery failure of the malware host. In the experiment environment, only the simplest HTTP server is set up to answer the malware acquisition calls. Due to simplicity nature, HTTP request-dropping events occurred occasionally. This problem can be resolved by reevaluating the failed shellcode samples or deploying a malware host that is more reliable. The second category only demonstrated the traits of malware acquisition behavior. This is a common problem of evaluating a malware sample on a dynamic analysis platform where execution time period expires before the malware is completely acquired from the malware host. To resolve the problem, a longer execution time period is assigned for another round of evaluation. The last category shows no sign of shellcode sample execution. After investigation, the error occurs during shellcode extraction process where the shellcode code segment exists before the location of the GetPC code [20]. In other words, code segment exists before the detected shellcode offset in shellcode extraction procedure. To resolve this problem, it requires human interpretation to correct shellcode offset and the entry point of the shellcode.

VII. LIMITATIONS AND FUTURE WORK

The proposed system does not guarantee the correctness of malware execution that exercises intended malicious behaviors. To ensure correctness of the intended malicious behaviors of malware is a difficult problem to be solved. Instead, we take the approach in evaluation to ensure that the targeted malware has been exercised, and our proposed scheme works properly according to the hypothesis.

One limitation is based on shellcode detection of different types of remote exploit. As mentioned in the introduction, different form of remote exploitation exists, e.g., document files with embedded macros/scripts that decrypt and execute shellcode. In such a scenario, the shellcode is usually encrypted where the shellcode detection heuristic used in this paper will produce a false negative result. Studies [31–34] showed that the embedded macro/script on the other hand is the medium for attacking the vulnerable functions, decrypting and executing the shellcode. Due to the scope of this paper, document malware of this kind is not included herein for the evaluation of effectiveness of our proposed scheme. However, our proposed scheme is also effective if the shellcode detection scheme is substituted with an appropriate document shellcode detection scheme and minor adjustments on the shellcode extraction scripts. If the extracted shellcode does not rely on the context of the document reader, the proposed scheme will work as it was originally proposed.

Shellcode execution that is strongly coupled with the context of the platform is another problem that can be addressed. This problem is well aware of in the beginning of this research. However, we have decided to address the context problem in future research work for the following reasons. For a piece of shellcode to remain portable, the developer will try to write a piece of shellcode that does not depend on the context of the platform. Therefore, a piece of portable shellcode can be adapted to a number of exploits. Thus, there is a good population for portable shellcode in the wild. Another obstacle is based on the malware sample collection. Since the malware sample is collected by Nepenthes passive honeypot and the vulnerabilities are emulated. It is reasonable to assume that the collected malware samples will have minimal dependency on the vulnerable services.

VIII. CONCLUSION

In this paper, we proposed a new scheme enabling dynamic malware analysis to analyze malware where the targeted malware requires supplementary execution information. The scheme leverages the results of honeypot/intrusion detection research of shellcode detection to extract shellcode to aid the dynamic malware analysis platform. As a result, the extracted shellcode will act as an agent that executes the target malware for dynamic malware analysis. The scheme is evaluated with wild malware captured by Dionaea honeypot platform and 97% of the malware samples demonstrated execution behaviors when the proposed scheme is applied. Additionally, a formal model is constructed to support the proposed scheme. Using our proposed scheme, information security experts can instantly analyze a piece of malware that requires supplementary execution information without spending additional time to find out the execution conditions. Furthermore, the propose scheme is automated to save human effort in generating the execution conditions for malware.

ACKNOWLEGMENTS

This work was supported in part by MOST 104-2218-E-001-002, Taiwan Information Security Center (TWISC), Ministry of Science and Technology, Ministry of Education (R.O.C.), Bureau of Investigation, National Security Council of Taiwan, TTC, NCC, Chunghwa Telecomm., TSMC, and Trend Micro Inc., respectively.

REFERENCES

[1] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation," in *Proc. of 16th USENIX Security Symp.*, 2007, pp. 167-182.

[2] I. Arce, "The Shellcode Generation," in *Security & Privacy 2.5*, IEEE, 2004, pp. 72-76.

[3] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni, *Metasploit: the penetration tester's guide*. San Francisco, CA: No Starch Press, 2011.

[4] Meatsploit Development Team, (2014), *Metasploit Project* [Online]. http://www metasploit.com.

[5] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling, "The Nepenthes Platform: An Efficient Approach to Collect Malware," in *Recent Advances in Intrusion Detection (RAID)*, 2006, pp. 165-184.

[6] M. Sikorski, and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. San Francisco, CA: No Starch Press, 2012.

[7] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," in *Journal of IEEE Security and Privacy 5.2*, March 2007, pp. 32-39.

[8] U. Bayer, C. Kruegel, and E. Kirda, "TTAnalyze: A Tool for Analyzing Malware." na, 2006.

[9] C. Guarnieri, et al., (2014), *Cuckoo Sandbox* [Online]. http://www.cuckoosandbox.org.

[10] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," in *USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, April 2009.

[11] D. Brumley, et al., "Automatically Identifying Trigger-based Behavior in Malware," in *Botnet Detection*. Springer US, 2008, pp. 65-88.

[12] A. Moser, C. Kruegel, and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," in *Proc. of the 2007 IEEE Symp. on Security and Privacy (SP '07)*, 2007, pp. 231-245.

[13] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware-analysis Techniques and Tools," in *ACM Computing Surveys 44.2:6 (CSUR)*, February 2012.

[14] J. C. King, "Symbolic Execution and Program Testing," in *Communications of the ACM 19.7*, July 1976, pp. 385-394.

[15] K. Sen, D. Marinov, and G. Agha, "CUTE: a Concolic Unit Testing Engine for C", in *the Proc. of the 10th European Software Engineering Conference (ESEC/FSE-13)*, 2005, pp. 263-273.

[16] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, "Comprehensive Shellcode Detection using Runtime Heuristics," in *Proc. of the 26th Annu. Computer Security Applications Conf. (ACSAC '10)*, 2010, pp. 287-296.

[17] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos, "Emulation-based Detection of Non-self-contained Polymorphic Shellcode," in *Recent Advances in Intrusion Detection (RAID)*, 2007, pp. 87-106.

[18] Q. Zhang, D. S. Reeves, P. Ning, and S. P. Iyer, "Analyzing Network Traffic to Detect Self-decrypting Exploit Code," in *Proc. of the 2nd ACM Symp. on Information, Computer and Communications Security (ASIACCS '07)*, March 2007, pp. 4-12.

[19] P. Baecher and M. Koetter, (2014), *libemu* [Online]. http://libemu.carnivore.it.

[20] SkyLined and Cipher, (2014), *Hacking/Shellcode/GetPC* [Online]. http://skypher.com/wiki/index.php?title=Hacking/Shellcode/GetPC.

[21] C. Anley, J. Heasman, F. Linder, and G. Richarte, *The Shellcoder's Handbook: Discovering Exploiting Security Holes 2nd Edition*. Indianapolis: Wiley, 2007.

[22] Skape, (2003), *Understanding Windows Shellcode* [Online]. http://www hick.org/code/skape/papers/win32-shellcode.pdf.

[23] SkyLined and Cipher, (2014), *Hacking/Shellcode/kernel32* [Online]. http://skypher.com/wiki/index.php?title=Hacking/Shellcode/kernel32.

[24] Dionaea Development Team, (2014), *Dionaea Catches Bug* [Online]. http://dionaea.carnivore.it/.

[25] Microsoft Security TechCenter, (2014), *Microsoft Security Bulletin MS08-067 – Critical* [Online]. https://technet microsoft.com/en-us/library/security/ms08-067.aspx.

[26] Dionaea Development Team, (2014), *MS10-061 Attacks?* [Online]. http://carnivore.it/2010/10/18/ms10-061_attacks.

[27] Microsoft Security TechCenter, (2014), *Microsoft Security Bulletin MS10-061 – Critical* [Online]. https://technet microsoft.com/en-us/library/security/ms10-061.aspx.

[28] C. W. Wang, C. W. Wang, C. W. Hsu, and S. P. Shieh, "Malware Behavior Analysis Based on Virtual Machine Introspection and Snapshot Comparison," in *the 20th Cryptology and Information Security Conference (CISC 2010)*, Taiwan, May 2010, pp. 69-74.

[29] C. W. Wang, and S. P. Shieh, "SWIFT: Decoupled System-Wide Information Flow Tracking and its Optimizations," in *Journal of Information Science and Engineering*, 31.4, 2015, pp. 1413 – 1429.

[30] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *USENIX Annu. Technical Conf.*, FREENIX Track, April 2005, pp. 41-46.

[31] D. Stevens, “Malicious PDF Documents Explained,” in *Security & Privacy 9.1*, IEEE, 2011, pp. 80-82.

[32] Z. Tzermias, G. Sykiotakis, M. Polychronakis, and E. P. Markatos, “Combining Static and Dynamic Analysis for the Detection of Malicious Documents,” in *Proc. of the Fourth Enropean Workshop on System Security (EUROSEC ’11)*, 2011.

[33] R. Merritt, (2014), *Analyzing PDF Malware – Part 1* [Online]. http://blog.spiderlabs.com/2011/09/analyzing-pdf-malware-part-1.html.

[34] R. Merritt, (2014), *Analyzing PDF Malware – Part 2* [Online]. http://blog.spiderlabs.com/2012/01/analyzing-pdf-malware-part-2.html.

AUTHOR BIOGRAPHY

Michael Cheng Yi Cho is a Ph.D. candidate in the Department of Computer Science at National Chiao Tung University, Hsinchu, Taiwan. His research interests include Named Data Networking, access control, network security, and malware analysis.

Zhi-Kai Zhang is a Ph.D. candidate in the Department of Computer Science at National Chiao Tung University, Hsinchu, Taiwan. His research interests include cryptography, IoT security, cloud security, and access control.

Prof. Shiuhpyng Winston Shieh received his M.S. and Ph.D. degrees in electrical and computer engineering from the University of Maryland, College Park, respectively. He is a Distinguished Professor of Computer Science Department and the Director of Taiwan Information Security Center at National Chiao Tung University (NCTU). He has served as the adviser to the National Security Council of Taiwan, the chair of the Department of Computer Science, NCTU, and President of Chinese Cryptology and Information Security Association (CCISA). Being actively involved in IEEE, he has served as EIC of IEEE Reliability Magazine, EIC of RS Newsletter, Reliability Society VP Tech, Editor of IEEE Trans. on Reliability and IEEE Trans. on Dependable and Secure Computing. Dr. Shieh has also served as ACM SIGSAC Awards Committee member, Associate Editor of ACM Trans on Information and System Security, Journal of Computer Security, Journal of Information Science and Engineering, Journal of Computers, and the guest editor of IEEE Internet Computing, respectively. Furthermore, he has been on the organizing committees of many conferences, such as the founding Steering Committee Chair and Program Chair of ACM Symposium on Information, Computer and Communications Security (AsiaCCS), founding Steering Committee Chair of IEEE Conference on Dependable and Secure Computing, Program Chair of IEEE Conference on Security and Reliability. Along with Virgil Gligor of Carnegie Mellon University, he invented the first US patent in the intrusion detection field, and has published 200 technical papers, patents, and books. He is an IEEE Fellow, ACM Distinguished Scientist, and Distinguished Pro-fessor of Chinese Institute of Electrical Engineers. His re-search interests include system penetration and protection, malware behavior analysis, network and system security.