Towards Automated Characterization of Malware's High-level Mechanism using Virtual Machine Introspection

Shun Yonamine[1], Youki Kadobayashi[1], Daisuke Miyamoto[2] and Yuzo Taenaka[1]
[1]*Nara Institute of Science and Technology, 8916-5 Takayama, Ikoma, Nara 630-0192, Japan*
[2]*The University of Tokyo, 2-11-16 Yayoi, Bunkyo, Tokyo, 113-8658, Japan*

Keywords: Malware Characterization, Virtual Machine Introspection, Taint Analysis, Malware Analysis.

Abstract: One of the goals of malware analysis is to figure out the intention of an attacker, namely high-level mechanism. Since malicious activities are typically performed by combining multiple APIs, to identify the malicious intention, it is needed to inspect the series of APIs to analyze its semantics. In traditional malware analysis, this task generally relies on manual efforts of experts. There is no methodology for associating multiple APIs and identifying the malicious intention in an automated manner. In this paper, we propose a virtual machine introspection-based method for automatically identifying high-level mechanisms. We developed Spaniel, a prototype system, which uses taint analysis to track malicious processing that derives from the data read from a specified file and collects the traces of malicious activities. For evaluation, we used adversary behavior models defined in ATT&CK and Spaniel identified key indicators that cover 26% of those models.

1 INTRODUCTION

One of the goals of malware analysis is to understand the intention of an attacker. Since malware performs malicious activities following the intention of an attacker, security analysts need to figure out malicious activities, namely high-level mechanism (Mitre, 2018; Lee et al., 2013). Malicious activities are conducted through the series of low-level actions such as system calls. And then, the middle-level behavior can be retrieved from a series of low-level actions. For instance, given a middle-level behavior that it first calls `read()` to access user's password file and next calls `send()` to send that data outside the network, there is a possibility that it is intended for data theft. Therefore, in malware analysis, it is important to understand the relationship of each system call that was executed independently by malware.

In traditional malware analysis, security analysts monitor APIs and/or system calls executed by malware. API monitoring is one of the common malware analysis techniques (Egele et al., 2012). Although API monitoring enables security analysts to collect rich information about malware, it is extremely time-consuming since it requires analysts many steps, such as setting breakpoints, inspecting memory values, etc. There have been analysis platforms that support automated malware analysis, e.g., Cuckoo Sandbox (Cuckoo, 2013) . However, the information shown by those platforms are limited and they do not preserve the details about the relationship of each API call.

In this paper, we propose a novel approach to associate multiple actions of malware and extract its behavioral characteristics. Our approach aims at identifying the malicious mechanisms of malware by analyzing its runtime behavior and reconstructing its semantics. To accomplish this, we leverage technologies of virtual machine introspection and taint analysis. This paper presents Spaniel, a prototype system for automatically extracting relationships of each low-level action executed by malware. Our approach is based on the insight that most types of malicious activities accompany manipulation to the file data. Spaniel performs taint analysis against file data and associates low-level actions with each other through tainting.

In order to show the capability of our approach, we conduct a series of intention analysis experiment including the scenarios of Exfiltration, Command and Control (C2), and Encryption. Further, we investigate the applicable coverage of our approach in the analysis of malicious activities. We used the adversary's behavior model defined in ATT&CK (Strom et al., 2017; Mitre, 2018) matrix which consists of various techniques commonly used by an attacker.

Yonamine, S., Kadobayashi, Y., Miyamoto, D. and Taenaka, Y.
Towards Automated Characterization of Malware's High-level Mechanism using Virtual Machine Introspection.
DOI: 10.5220/0007405504710478
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 471-478
ISBN: 978-989-758-359-9
Copyright © 2019 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

2 RELATED WORK

The automated methodologies for analyzing the malicious activity of malware is a widely studied topic. However, the way of leveraging dynamic analysis techniques to analyze malicious intentions has not been widely studied yet. Existing methods are tailored to solve a specific problem in malware analysis. In order to detect a characteristic of spyware, the information flow tracking method was taken (Yin et al., 2007; Egele et al., 2007). Further, information flow tracking-based methods to analyze the cryptographic function of malware were proposed (Wang et al., 2009; Gröbert et al., 2011). Jacob et al. (Jacob et al., 2011) proposed a program analysis based method to identify C2 communication of bot. As for the analysis of code injection malware, the methods of process tracking were studied (Caillat et al., 2015; Korczynski and Yin, 2017). Many of those previous researches share the key insight that how a program processes a message gives rich information about the behavioral characteristic of malware. In this paper, we apply this insight to build a method for identifying the intention of an attacker.

Also, the techniques that can be leveraged for automated malware analysis is widely studied. The virtual machine introspection (VMI) is a technology for inspecting a system running on the virtual machine from outside the hypervisor (Garfinkel and Rosenblum, 2003) and enables whole-system dynamic malware analysis. VMI is widely used for many security solutions (Dolan-Gavitt et al., 2011) , e.g., intrusion detection, forensics, malware analysis. There are several projects that feature VMI technology, such as DRAKVUF (Lengyel et al., 2014), PANDA (Panda-re, 2018; Dolan-Gavitt et al., 2015) , and DECAF (Henderson et al., 2014). Furthermore, VMI platforms that are based on QEMU is widely used for dynamic taint analysis (Schwartz et al., 2010) . Dynamic taint analysis (taint analysis) leverages dynamic binary instrumentation (DBI) technology, and then it enables data tracking in instruction-level. Taint analysis can be used in malware analysis for analyzing specific functionalities of malware(Yin et al., 2007; Wang et al., 2009).

To perform malware analysis effectively, there is a study about whole-system dynamic binary analysis approach. The whole-system dynamic binary analysis is a technique used for analyzing malicious code by using the virtual machine. Although developing whole-system dynamic binary analysis tool from scratch is not straightforward, recent studies (Henderson et al., 2014; Dolan-Gavitt et al., 2015) have developed platforms to facilitate those whole-system dynamic binary analysis techniques. We also leveraged those efforts to develop our proposed method.

3 SYSTEM DESIGN

3.1 Behavioral Analysis Method based on File-monitoring and Tainting

The key feature in our approach is using the data flow to automatically extract every APIs that are associated with each other. Our approach aims at retrieving the profile of middle-level behavior needed to reason the intention of an attacker. To accomplish this, our approach uses API monitoring and taint analysis (tainting) based on VMI.

First, before staring malware analysis, we have to specify the watched file, a file which can be a data source for tracking data flow. The watched file is used as the taint source for taint analysis. For instance, when it detects the `read()`API to the watched file, it launches taint analysis on the memory where the file data is loaded. We track the whole of the file data at the byte level. By tracking the propagation of taint, it can be possible to extract the series of instructions that relate to each other. We collect tainted instructions, the code of instructions that processed tainted data. From tainted instructions, also API calls that handled tainted data can be retrieved.

Further, it makes possible to detect the presence of an attacker's intention by using the traces of taint as the indicator. The traces of taint can be retrieved by taint check, checking if the memory or register handled by a tainted instruction is tainted or not. For instance, the traces of taint could indicate the data exfiltration if `send()`API handle tainted data on its buffer. Moreover, our approach provides a visualization of an analysis result that shows the relationship between each low-level action. The visualization is designed to aid security analysts to estimate the malicious intention of an attacker.

3.2 Implementation

We developed Spaniel, a prototype of our proposed system. Spaniel is a plugin for PANDA (Panda-re, 2018; Dolan-Gavitt et al., 2015). PANDA is a whole-system dynamic binary analysis platform that supports record-and-replay based analysis. Record-and-replay can decouple the analysis from the execution and thus suited for taint analysis that is too expensive to be applied at runtime.(Chow et al., 2008; Stamatogiannakis et al., 2015). As for taint analysis, in

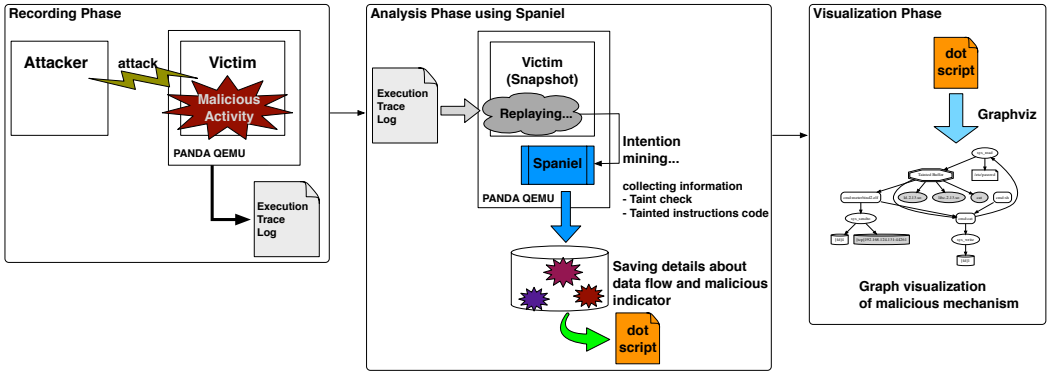

Figure 1: System overview.

our case, we leverage the implementation of PANDA (Whelan et al., 2013) to monitor the propagation of taint at the byte level. PANDA leverages QEMU to instrument program execution per emulated code. Thus, we can monitor and instrument every execution on the system per emulated code to track the data flow. Additionally, in our use of PANDA, we enabled the taint propagation through pointer dereference in order to capture the data flow on the malicious activity in detail.

3.3 The Procedure of Malware Analysis in our Approach

Spaniel performs analysis of malicious activities using record-and-replay. In addition, Spaniel has three phases in analyzing malware as shown in Figure 1: the recording phase for obtaining execution trace of malware, the analysis phase for collecting information used to identify the presence of malicious activity, and the visualization phase for making a visualized output of analysis result.

In the recording phase, we execute a malware in a sandboxed environment and record its execution to obtain execution trace log. Execution trace log must be captured for malware analysis using Spaniel. During malware performs its malicious activity, PANDA records execution trace. Our assumption is that malware thoroughly performs its malicious activity during the recording phase.

In the analysis phase, we use Spaniel to analyze adversary scenario from execution trace that we recorded. While observing replayed malicious activity, Spaniel performs API hooking to monitor the read access to the watched file that we specified. If our watched file is accessed by file read operation, Spaniel performs tainting on read-buffer and starts taint analysis. Spaniel then collects the tainted instruction code which handled the tainted data and caused taint propagation. Further, Spaniel uses VMI to obtain names of shared libraries that were referenced from tainted instruction codes. Spaniel also hooks output-related API (e.g., `write()`, `send()`) to conduct taint check on write-buffer. If the buffer is tainted, under the policies we defined, we consider it as an indicator of data theft or data tampering. In this phase, we collect information for identifying the type of malicious attempt.

In the visualization phase, Spaniel generates a graph that represents the series of malicious actions that are associated with our watched file data. Based on the analysis result, Spaniel produces a dot script used by Graphviz (Graphviz, 2018) to output a graph. The visualization phase is designed to help intuitively understand the analysis results, for a case of reasoning about malicious activities that are not yet defined under our detection policies.

As a result of three phases, Spaniel obtains the indicator of adversary intent from recorded malware's activities. We use this as the indicator of compromise (IOC) as well as the evidence that identifies the type of high-level mechanism.

4 EXPERIMENT

In this section, we test Spaniel on its capability of identifying the attacker's intention. We demonstrate that Spaniel can analyze footprints of malicious actions and associate them to identify malicious characteristics. We set up experiments of three case studies namely, Exfiltration, Encryption, Command and Control (C2) all selected from ATT&CK (Mitre, 2018). Since Spaniel performs malware analysis through replaying the execution trace log, we need to run malware samples on virtual machine and record malicious activities in advance. Malicious activities are recorded using the record-and-replay functionality of

PANDA and saved into execution trace log as shown in Figure 1.

In order to set up those experiments, we used Linux as a victim's platform and prepared malware samples that are appropriate for each experiment. We used meterpreter (Security, 2018) to simulate cases of exfiltration and C2. In addition, we used OpenSSL (Foundation, 2018) as a sample for simulating encryption.

We set up experiments as follows. In the cases of exfiltration and C2, we simulate adversary behavior accessing victim machine via meterpreter. In these scenarios, the attacker attempts to steal a credentials file (`/etc/passwd`) using `cat` command via meterpreter and sends it outside the network. In the case of encryption, Spaniel analyzes the execution trace log that records encryption processing performed by OpenSSL that we imitate as ransomware. In this scenario, ransomware targets simple text file (`cryptme.txt`). In all of our experiments, we employ a strategy of monitoring the system calls that access our watched file (e.g. `/etc/passwd`, `cryptme.txt`). Spaniel implements this strategy to start taint analysis when it detects the read-related APIs to our watched file.

4.1 Detecting IOC of "Exfiltration Over C2 Channel"

We demonstrate that Spaniel analyzes "Exfiltration Over C2 Channel" attack model having both aspects of exfiltration and C2. We created a scenario where the attacker tries to steal credentials file with `cat` command via meterpreter. Spaniel then analyzes the execution trace log which recorded the attack we simulated. The goal of this experiment is to detect indicators that identify each attack of exfiltration and C2.

4.1.1 The Case of Exfiltration

In order to find the essence of the high-level mechanism of file exfiltration activity, before network transfer, we have to detect the send-buffer, which holds data tainted and associated with our watched file. Spaniel applies taint analysis to data of our watched file and tracks every instructions that handle tainted data through taint propagation. At the moment when network transfer happens, Spaniel conducts taint check to data held in the send-buffer. If the send-buffer is tainted, we regard it indicates a sign of occurrence the data exfiltration because we think it is anomalous that the taint tag associated with credentials data is propagated to the send-buffer.

In many cases, the modern malware performs

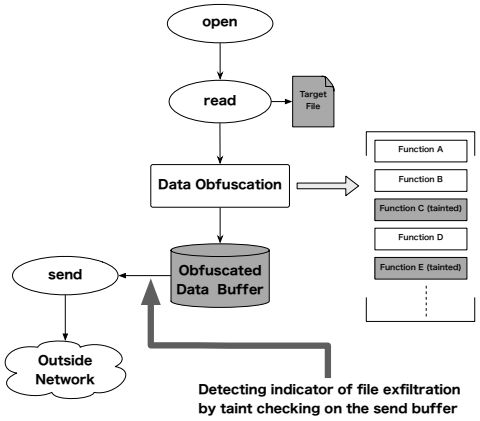

Figure 2: Detecting indicator of "File Exfiltration".

obfuscation or tampering against target data before sending them to the attacker's machine. Since the tampered data, which is encrypted or compressed, shows no signs of original data on its surface, then it becomes harder to associate the data kept in the send-buffer with stolen data. Therefore, we use taint checking to detect a sign of network transfer that targets our watched data, as shown in Figure 2. Through taint propagation, our watched data leaves its vestiges throughout memory areas or registers. Spaniel performs taint checking to check if buffers handled by APIs are tainted or not. If tainted, we consider that fact as an indicator of file exfiltration.

We verified our hypothesis in the analysis of meterpreter as follows. When `read()` API is called and it reads our watched file (`/etc/passwd`), then Spaniel applies taint analysis to read-buffer. In the subsequent processing after taint analysis is enabled, we observed `send()` API call holding tainted buffer on its argument. Consequently, we consider those tainted buffer as an indicator of file exfiltration.

4.1.2 The Case of Command and Control

We analyze meterpreter for the purpose of detecting the presence of command and control. Spaniel applies taint analysis to `/etc/passwd` in the same way we performed in Section 4.1.1 and collects tainted instructions. In dealing with a malware that implements C2, we assume that an attacker is likely to utilize system utilities, e.g., command and shared libraries, during malicious activities in C2 channel. If the attacker follows our assumption, tainted instructions are likely to contain much information about the traces of the attacker.

Spaniel instruments the execution to examine tainted instructions during the analysis. While collecting tainted instructions as shown in Figure 2,

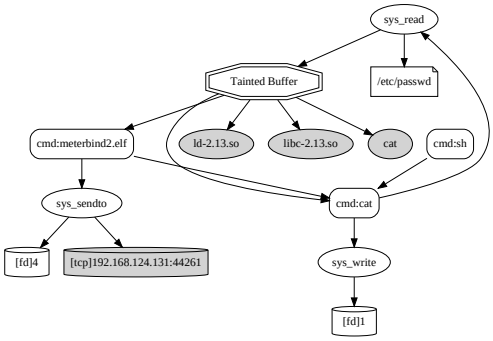

Figure 3: Visualization of "Exfiltration Over C2 Channel".

Spaniel leverages VMI to obtain a list of processes as well as shared libraries loaded for each process. We found out that some tainted instructions are codes that are used by `cat` command. Thus, we can confirm the presence of a malicious activity via the C2 channel.

Also, we found out that it is possible to associate exfiltration and C2 activities. We used the name of system utility as an indicator of the malicious attempt through C2. Some malware use system utilities to make it difficult to distinguish malicious operations and legitimate ones. However, we think examining tainted instructions helps analysts to deal with this problem.

4.1.3 Visualization of Execution Trace of "Meterpreter"

We visualized our experiment result as a graph (Figure 3). From a dot script that Spaniel generated from anaysis result, we can obtain a graph by using Graphviz. This graph shows a series of actions that were performed by malware to accomplish its malicious attempt of data theft. Each node represents a system component, e.g., process, system call, that we observed during record-and-replay based analysis with Spaniel. Edges between nodes represent caller/callee relationships. The "Tainted Buffer" node represents the buffer that holds the data of our watched file (`/etc/passwd`). Every node is involved in handling data that are tainted. We hope this graph helps analysts understand the mechanism of exfiltration and C2 more intuitively and deal with threats.

4.2 Detecting IOC of "Encryption"

4.2.1 The Case Encryption of OpenSSL

In this section, we demonstrate the way of detecting the evidence of the use of encryption from a malicious activity. In order to solve this task, we have to confirm

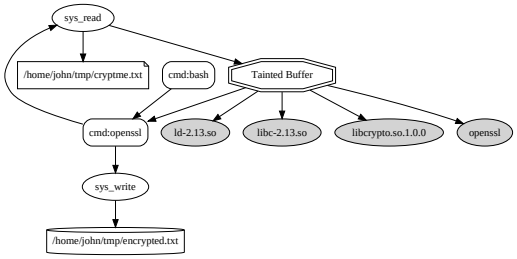

Figure 4: Visualization of "Encryption".

that the malware uses crypto-related API through the series of malicious actions. Thus we examine tainted instructions to find the traces of using crypto-related API. Since tainted instructions are instruction codes which derive from tainting our watched file data, we assume that tainted instruction codes give us more details about malicious processing.

We conducted our experiment as follows. First, we simulated encryption processing and obtained execution trace log. In this scenario, openssl command performs AES encryption against a simple text file, namely `cryptme.txt`. We next use Spaniel to analyze the encryption scenario that we simulated. Further, we try to collect the evidence of encryption from a significant amount of instruction codes.

The openssl command performs encryption processing to our watched file, `cryptme.txt`. In the similar way that we analyzed meterpreter in Section 4.1, Spaniel monitors the file operations and starts taint analysis when our watched file is accessed, and then collects tainted instructions. While collecting tainted instruction codes, Spaniel leverages VMI to obtain details of tainted instructions, such as addresses and names of shared libraries, like we have conducted to find a footprint of `cat` command in Section 4.1.2. From the experiment result, we confirmed that `libcrypto.so` was included in a list of shared libraries we obtained by using VMI. The `libcrypto.so` is a shared library used for encryption processing, and then we confirmed the possibility that our watched file data was processed with crypto-related APIs. From the experiment result, we confirmed that examing tainted instructions is efficient to find the traces of using the crypto-related libraries and identify the malicious mechanism of encryption.

4.2.2 Visualization of Execution Trace of "Openssl"

In the same way that we visualized exfiltration and C2, we obtained a graph (Figure 4) that represents file encryption activity. This graph shows the relationship between each node of handling tainted data

and follows the same representation rule that we used in Section 4.1. This graph shows `read()` API call, which handles our watched file `cryptme.txt`. The "Tainted Buffer" node represents data of our watched file. Further, the `libcrypto.so` node indicates that our file data is encrypted. From the name of crypto-related API we obtained, we can identify the malicious attempts of encryption.

5 EVALUATION

In order to confirm the effectiveness of Spaniel, we investigated its capability of finding IOCs that we expect to find from various kinds of malicious activities listed in ATT&CK. ATT&CK matrix includes 106 models of malicious activities. First, we explain our IOC for identifying the kinds of malicious activities as follows.

- If instruction codes executed by malware handle tainted data.
- If tainted data is stored on the argument when output-related API is called.
- If it is possible to detect what kind of activity is performed by malware from the name of the shared library mapped from instruction code. e.g., We identified the occurrence of encryption by detecting API of `libcrypto.so` in an experiment.

Next, we conducted an investigation and the results can be seen in Table 1. This result shows that Spaniel is capable of finding the IOC that we described above from 26 models out of a total of 106 models. This result indicates that Spaniel is potentially able to extract the middle-level behavior of the malicious activity on every stage of the cyber kill chain, (e.g., Control, Execute, Maintain). Those models Spaniel could detect have a common behavioral characteristic; the file input and the output to external resources such as a file or socket, can be easily related by taint propagation and taint checking. We demonstrated that Spaniel could detect models where explicitly data flow by data alteration (e.g., encryption) occurs between the file input and external output. We want to state that, in the security incident caused by malware, the analysis method based on tracking the file data accessed by malware is a reasonable approach.

We examined "Data source" item from each technique of ATT&CK (Mitre, 2018). "Data source" contains information that can be used to detect and analyze each attack model. Since Spaniel performs taint analysis against malware's file operations, we enumerated models that have "File monitoring" in their data sources, namely file-monitoring type. There are 56 models of file-monitoring type. We confirmed that there are 18 attack models out of 56. However, that is fewer than the 26 models shown in Table 1. We note several reasons as follows.

- "Account discovery" model of discovery tactic is not defined as a file-monitoring type. Regarding "Account discovery", ATT&CK considers only process activities, e.g., id command and groups command, in data sources. However, we confirmed it empirically through the experiment of exfiltration targeted at `/etc/passwd`.
- "Exfiltration over command and control channel" model of exfiltration tactic is not defined as the file-monitoring type. Regarding this technique, ATT&CK considers data sources only about process activities and network activities.
- Several models of C2 tactic are not defined as the file-monitoring type. Regarding these techniques, ATT&CK considers their data sources as network activities such as packet monitoring.

In the descriptions of ATT&CK, to analyze a malware's network activity, packet monitoring is regarded as an appropriate method. This also means that network traffic data is generally considered as an appropriate data source for malware analysis.

We expect that, since the data source is limited to only a regular file, the analysis enabled coverage of Spaniel is also limited. From this, we want to point out that there is a gap between the traditional malware analysis method and the data flow tracking-based method. We suppose that considering the kind of data source, e.g., file or network, is important since the kind of data source which malware tries to access varies depending on its purpose.

Although the result was lower than half the total number, we are not pessimistic about this result. We evaluated through a series of experiments (e.g., exfiltration, encryption, C2), the policy we used for evaluation might not be severe. For instance, we did not consider the case of "Automated Collection" model into the results even though we accomplished an experiment of credentials file exfiltration in Section 4.1 since "Automated Collection" belongs to collection tactics, not exfiltration tactics. On the malware analysis that uses the data flow tracking, it is needed to design appropriate policies to determine if the behavior tracked through data flow is malicious or not. Depending on detection policies, we possibly could improve detection rate against ATT&CK models.

Table 1: A list of attack models in the ATT&CK Matrix that Spaniel can detect.

Tactic	Technique	Detected attack models as ◎, similar models as ○
Persistence	.bash_profile and .bashrc	◎
	Hidden Files and Directories	◎
	Rc.common	○
Privilege Escalation	Setuid and Setgid	◎
Defense Evasion	Clear Command History	○
	File Deletion	◎
	Hidden Files and Directories	◎
	Scripting	○
Credential Access	Bash History	○
	Credentials in Files	◎
Discovery	Account Discovery	○
Lateral Movement	Remote File Copy	◎
Execution	Command-Line Interface	◎
	Scripting	○
Collection	Data Staged	○
	Data from Local System	○
Exfiltration	Data Compressed	○
	Data Encrypted	◎
	Exfiltration Over Command and Control Channel	◎
Command and Control	Commonly Used Port	○
	Custom Command and Control Protocol	○
	Custom Cryptographic Protocol	◎
	Data Encoding	◎
	Data Obfuscation	◎
	Remote File Copy	◎
	Standard Cryptographic Protocol	○

6 DISCUSSION

In this section, we discuss the limitations in Spaniel and suggestions for future work. First, Spaniel is designed to handle malware that does not use techniques to thwart analysis using a virtual machine. If malware detects the presence of virtual machine and then stops or changes its behavior, recording phase (Figure 1 in Section 3) may not work effectively. Since Spaniel relies on record-and-replay using QEMU, we need countermeasures on each anti-analysis technique against QEMU. It still leaves a technical challenge to deal with real-world malware that uses those anti-analysis techniques

Next, the predefined adversary behavior models of ATT&CK we detected are explicit data processing and their maliciousness could only be identified from the relationship between file input and external output, e.g., file or network, through tainting. However, also taint analysis has the weakness, e.g., overtainting or undertainting (Schwartz et al., 2010; Slowinska and Bos, 2009) . We have to consider possibilities where malware generates intentionally indirect or implicit data flow to bypass taint analysis.

Next, Spaniel detects encryption activities of the malware based on the presence of crypto-related API calling. Therefore, Spaniel can be evaded if malware performs encryption with its own method. Our approach is a sort of signature-based detection. We treated the crypto-related APIs as an explicit indicator for detection. To deal with encryption activity that does not match with signatures, it is needed to adopt the heuristic-based approaches. However, also heuristic-based (Wang et al., 2009; Gröbert et al., 2011) approaches rely on assumptions that are statistical and/or empirical. If malware does not follow those assumptions, also heuristic-based approaches can be bypassed. We leave this to future work.

Finally, although we expect that Spaniel can help security analysts in the complicated tasks of malware analysis, we have not tested Spaniel from a perspective of performance improvement of analysts. To evaluate efficiency from a view of security analysts, we might need to conduct user tests. For example, Yakdan et al. (Yakdan et al., 2016) have conducted a user study for evaluating the usability of decompiler they designed to help reverse engineers.

7 CONCLUSION

Understanding the attacker's intention is one of the challenges in malware analysis. From the perspective of automated malware analysis, there is no method to reason about the intention of an attacker. In this paper, we proposed a novel approach to pinpoint the kind of malicious mechanism. Spaniel, a prototype we developed, examines instruction codes that relate to the file operations by using taint analysis. In order to confirm our hypothesis, we tested Spaniel with several attack models, exfiltration, encryption, C2. We confirmed that Spaniel is capable of detecting IOC and identifying the type of high-level mechanism. Through the series of experiments, we used minimal-installed Linux as victim's system and time cost for analysis was less than 5 minutes. We hope this type of characterization method would give more insights in this field of malware analysis.

REFERENCES

Caillat, B., Gilbert, B., Kemmerer, R., Kruegel, C., and Vigna, G. (2015). Prison: Tracking process interactions to contain malware. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on*, pages 1282–1291. IEEE.

Chow, J., Garfinkel, T., and Chen, P. M. (2008). Decoupling dynamic program analysis from execution in virtual environments. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 1–14.

Cuckoo (2013). Automated Malware Analysis. https://www.cuckoosandbox.org/.

Dolan-Gavitt, B., Hodosh, J., Hulin, P., Leek, T., and Whelan, R. (2015). Repeatable reverse engineering with panda. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, page 4. ACM.

Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J. T., and Lee, W. (2011). Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *IEEE Symposium on Security and Privacy*, pages 297–312. IEEE Computer Society.

Egele, M., Kruegel, C., Kirda, E., Yin, H., and Song, D. (2007). Dynamic spyware analysis.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, 44(2):6.

Foundation, O. S. (2018). OpenSSL Cryptography and SSL/TLS Toolkit. https://www.openssl.org/.

Garfinkel, T. and Rosenblum, M. (2003). A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium*.

Graphviz (2018). "Graphviz - Graph Visualization Software". https://www.graphviz.org/.

Gröbert, F., Willems, C., and Holz, T. (2011). Automated identification of cryptographic primitives in binary programs. In *International Workshop on Recent Advances in Intrusion Detection*, pages 41–60. Springer.

Henderson, A., Prakash, A., Yan, L. K., Hu, X., Wang, X., Zhou, R., and Yin, H. (2014). Make it work, make it right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 248–258. ACM.

Jacob, G., Hund, R., Kruegel, C., and Holz, T. (2011). Jackstraws: Picking command and control connections from bot traffic. In *USENIX Security Symposium*, volume 2011. San Francisco, CA, USA.

Korczynski, D. and Yin, H. (2017). Capturing malware propagations with code injections and code-reuse attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1691–1708, New York, NY, USA. ACM.

Lee, A., Varadharajan, V., and Tupakula, U. (2013). On malware characterization and attack classification. In *Proceedings of the First Australasian Web Conference-Volume 144*, pages 43–47. Australian Computer Society, Inc.

Lengyel, T. K., Maresca, S., Payne, B. D., Webster, G. D., Vogl, S., and Kiayias, A. (2014). Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system. In *Proceedings of the 30th Annual Computer Security Applications Conference*.

Mitre (2018). "ATT&CK Linux Technique Matrix". https://attack.mitre.org/wiki/Linux_Technique_Matrix (accessed 2018-02-13).

Mitre (2018). "MAEC Core Specification, Version 5.0". http://maecproject.github.io/releases/5.0/MAEC_Core_Specification.pdf.

Panda-re (2018). "Platform for Architecture-Neutral Dynamic Analysis". https://github.com/panda-re/panda.

Schwartz, E. J., Avgerinos, T., and Brumley, D. (2010). All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and privacy (SP), 2010 IEEE symposium on*, pages 317–331. IEEE.

Security, O. (2018). About the Metasploit Meterpreter. https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/.

Slowinska, A. and Bos, H. (2009). Pointless tainting?: evaluating the practicality of pointer tainting. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 61–74. ACM.

Stamatogiannakis, M., Groth, P., Bos, H., et al. (2015). Decoupling provenance capture and analysis from execution. In *Proceedings of the 7th USENIX Workshop on the Theory and Practice on Provenance (TaPP). Edinburgh, Scotland*.

Strom, B. E., Battaglia, J. A., Kemmerer, M. S., Kupersanin, W., Miller, D. P., Wampler, C., Whitley, S. M., and Wolf, R. D. (2017). Finding cyber threats with att&ck-based analytics.

Wang, Z., Jiang, X., Cui, W., Wang, X., and Grace, M. (2009). Reformat: Automatic reverse engineering of encrypted messages. In *ESORICS*, volume 9, pages 200–215. Springer.

Whelan, R., Leek, T., and Kaeli, D. (2013). Architecture-independent dynamic information flow tracking. In *International Conference on Compiler Construction*, pages 144–163. Springer.

Yakdan, K., Dechand, S., Gerhards-Padilla, E., and Smith, M. (2016). Helping johnny to analyze malware: A usability-optimized decompiler and malware analysis user study. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 158–177. IEEE.

Yin, H., Song, D., Egele, M., Kruegel, C., and Kirda, E. (2007). Panorama: capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127. ACM.