Information Sciences 231 (2013) 45–63

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

In-execution dynamic malware analysis and detection by mining information in process control blocks of Linux OS

Farrukh Shahzad *, M. Shahzad, Muddassar Farooq

Next Generation Intelligent Networks Research Center (nexGIN RC), FAST National University of Computer and Emerging Sciences (FAST-NU), Islamabad, Pakistan

ARTICLE INFO

Article history:
Available online 29 September 2011

Keywords:
Intrusion detection system
Kernel task structure
Malware forensic
Operating system security
Malicious process detection

ABSTRACT

Run-time behavior of processes – running on an end-host – is being actively used to dynamically detect malware. Most of these detection schemes build model of run-time behavior of a process on the basis of its data flow and/or sequence of system calls. These novel techniques have shown promising results but an efficient and effective technique must meet the following performance metrics: (1) high detection accuracy, (2) low false alarm rate, (3) small detection time, and (4) the technique should be resilient to run-time evasion attempts.

To meet these challenges, a novel concept of *genetic footprint* is proposed, by mining the information in the kernel process control blocks (PCB) of a process, that can be used to detect malicious processes at run time. The *genetic footprint* consists of selected parameters – maintained inside the PCB of a kernel for each running process – that define the semantics and behavior of an executing process. A systematic forensic study of the execution traces of benign and malware processes is performed to identify discriminatory parameters of a PCB (`task_struct` is PCB in case of Linux OS). As a result, 16 out of 118 task structure parameters are short listed using the time series analysis. A statistical analysis is done to corroborate the features of the *genetic footprint* and to select suitable machine learning classifiers to detect malware.

The scheme has been evaluated on a dataset that consists of 105 benign processes and 114 recently collected malware processes for Linux. The results of experiments show that the presented scheme achieves a detection accuracy of 96% with 0% false alarm rate in less than 100 ms of the start of a malicious activity. Last but not least, the presented technique utilizes partial knowledge that is available at a given time while the process is still executing; as a result, the kernel of OS can devise mitigation strategies. It is also shown that the presented technique is robust to well known run-time evasion attempts.

1. Introduction

Computer malware is becoming an increasingly significant threat to the computer systems and networks world-wide. In recent years, security experts have observed a massive increase in the number and sophistication of new malware. According to a recent threat report by Symantec, in 2008 alone, 5491 new software vulnerabilities were reported, 1.6 million new malware signatures were created, 245 million new attacks were reported, and the financial losses caused by malware soared to more than 1 trillion dollars [14]. It is, however, interesting to note that though 50% of new malware are simply repacked

* Corresponding author.
E-mail addresses: farrukh.shahzad@nexginrc.org (F. Shahzad), muhammad.shahzad@nexginrc.org (M. Shahzad), muddassar.farooq@nexginrc.org (M. Farooq).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.09.016

versions of known malware [50], even then they successfully evade existing commercial-off-the-shelf antivirus software (COTS AV) because they utilize static signature-based techniques [49] that are not robust to code obfuscation and polymorphism.

To overcome shortcomings of existing COTS AV, malware researchers are focusing their attention to non-signature behavior based intelligent detection techniques that can detect new malware on zero day (at the time of its launch). Such techniques can be broadly classified into two categories: (1) static and (2) dynamic. The main objective of static behavior based techniques is to analyze the information and contents of a given file – contained in the header and payload – to build intelligent malware models and then use different classification engines to detect malware [44,45,47]. An inherent shortcoming of static techniques is to identify "robust" features that are difficult to evade by novel packing schemes. (Remember a packer only changes the contents of an executable without modifying its real semantics [12].)

In comparison, dynamic detection techniques try to model run-time behavior of a process; as a result, they cannot be evaded by mere code obfuscation or polymorphism. Two types of well known dynamic schemes have been proposed: (1) system call sequencing, and (2) flow graphs. The system call sequences based techniques (presented in [6,7,58,61,13,29]) build the behavioral model on the basis of the sequence of invoked system calls and the information passed in their arguments. These techniques have four well known shortcomings: (1) processing overhead of logging system calls, (2) high false alarm rate, (3) ability to make a decision only after analyzing the complete trace of the execution sequence of a process (by that time a malicious process has already done the damage), and (4) they can be easily evaded by simply reordering the system calls or adding irrelevant system calls to invalidate the sequence that represents malware.

The basic concept behind flow graph techniques is to construct taint graph of a running process by analyzing its data flow model [61,23]. The graph provides valuable insight about the activities of a running process. Most of the proposed schemes require a virtual machine with a shadow memory to keep track of taint information. A related system is NICKLE [40], but it has been demonstrated that return oriented rootkits [20] can evade it. These schemes – like their system call counterparts – have high false alarm rate, high processing overheads and high detection time (sometimes of the order of minutes). It is rightly concluded in a recent paper [24] that most of the above-mentioned dynamic techniques are novel and promising but still a leap jump is required in their detection accuracy, detection time, and processing overheads to establish their effectiveness to replace or complement traditional anti-virus software at an end host.

To meet these challenges, a novel technique based on dynamic analysis is proposed that uses *genetic footprint* of a running process. The thesis of using *genetic footprint* is: *the state diagram of malicious processes should be different from that of benign processes because of difference in their activities*. The *genetic footprint* is defined as a set of 16 out of 118 parameters, which are maintained by the kernel of an operating system, in the PCB of a process,[1] to keep track of the state of an executable process.[2] To be more specific, malicious processes that try to steal or corrupt data and information – like passwords, keystrokes, files etc. – try to covertly capture the information that is not intended for them [24] without a user's consent. In comparison, the benign processes follow access control policies to acquire the legitimate information. Similarly, the state of a malicious process that tries to replicate itself on storage media or network interfaces will be different from the benign process that runs only once, performs its given tasks and exits.

As mentioned before, the *genetic footprint* is maintained in the "task structure" by the kernel of every operating system. Linux operating system is selected because of its open source advantage that provides fine grain control over kernel structures; as a result, systematic studies/evaluations are conducted on them. In case of Linux, the information about the current state of a process is maintained in the `task_struct` structure. The aim is to analyze the temporal behavior of the *genetic footprint*; therefore, a system call is developed that dumps 118 fields of `task_struct` structure for 15 s with a resolution of 1 ms. This time interval is reconfigurable and at the moment it is based on the observation that most of the malicious processes in the selected dataset either finish their intended activity within this time duration or at least start showing a distinct behavior to characterize them as benign or malicious.

The proposed scheme is evaluated on a dataset that consists of 105 benign processes and 114 recently collected malware processes of Linux. The results of the experiments prove that proposed scheme achieves a detection accuracy of 96% with 0% false alarm rate. Moreover, it is able to detect a malicious process in less than 100 ms from "the start of a malicious activity". It is emphasized the use of "the start of a malicious activity" instead of "the start of a malicious process" because the use of *genetic footprint* of a process enables the detection of even those malicious processes which mostly behave like benign processes and perform malicious activity for a very short duration somewhere in the middle (such as backdoors). Most of existing run-time behavior analysis techniques fail to detect such malicious processes. Also to the best of our knowledge, no existing runtime analysis technique is able to detect a malicious activity in less than 100 ms. A robustness analysis of the proposed technique is also presented in circumstances when a crafty attacker splices the *genetic footprint* of a benign process with that of a malicious process at different positions. The results of experiments show that the proposed scheme is robust to such evasion attempts. It is also demonstrated that it is difficult to evade proposed technique if it is used in conjunction with recently proposed techniques – discussed in Section 6.

To conclude, the major contributions of the work presented in this paper are: (1) architecture of the proposed dynamic malware detection framework based upon the concept of *genetic footprint* – a set of selected parameters (maintained inside

[1] Throughout this paper, we will use the terms PCB, task structure, and `task_struct` interchangeably.

[2] The feasibility of using parameters of "task structure", maintained in the kernel of Linux, is investigated by the authors in a preliminary pilot study reported in [46].