International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-4, Issue-7)

Research Article

July 2015

Implementation of Malware Detector through Anomoly Detection Technique

Gousia Hazra Anjum Khan
M-Tech (C.S.E.) Persuing, ITMUR
Chhattisgarh, India

Kranti Kumar Dewangan
Assistant Professor (C.S.E.), ITMUR
Chhattisgarh, India

Abstract—

A malcode can be used for infilteration of hosts using a variety of methods,including attacks that exploit known software flaws,functionality hidden in regular programs and social engineering. The purpose of a malware detector is to identify the malware contents before it can effect or reach to the system or network. A brief study for the quality of the malware detector is to be taken into account for a wide variety of threats that malware poses.The malware is considered as a worldwide epidemic.As per the impact criteria malware impact is getting worse day by day.Thus,in our paper we aim to provide different malware detection tools i.e.malware detector including different techniques it uses.

Keywords— Malware, Signature-Based, Anomaly-Based, Malcode, Malware Detection, Malicious code.

I. INTRODUCTION

Security of data is one of the most important factors of information technology and communication.
[1.] Malware is a type of malicious software which interrupt the different operation on computer system, crashes the important information or private information of the computer system. In other words malicious software, malware refers to software programs designed to damage or do other unwanted actions on a computer system. As the name suggested malware, it can be broken down into mal-software which means software which performs mal functions, thus we can say it's a "bad ware". Malware is referred to by numerous names. Examples include malicious software, malicious code (MC) and malcode.

II. TYPES OF MALWARE

(i) Viruses: [2.] A computer virus is code that replicates by inserting itself into other programs. A program that a virus has inserted itself into is infected, and is referred to as the virus's host. An important caveat is that viruses, in order to function, require their hosts, that is, a virus needs an existing host program in order to cause harm. For example, in order to get into a computer system, a virus may attach itself to some software utility (e.g. a word processing application). Launching the word processing application could then activate the virus that may, for example, duplicate itself and disable malware detectors enabled on the computer system.
(ii)Worms: A computer worm replicates itself by executing its own code independent of any other program. The primary distinction between a virus and a worm is that a worm does not need a host to cause harm. Another distinction between viruses and worms is their propagation model. In general, viruses attempt to spread through programs/files on a single computer system. However, worms spread via network connections with the goal of infecting as many computer systems connected to the network as possible.
(iii) Trojan horses: A Trojan horse is malware embedded by its designer in an application or system. The application or system appears to perform some useful function (e.g., give the local weather), but is performing some unauthorized action (e.g., capturing the user's keystrokes and sending this information to a malicious host). Trojan horses are typically associated with accessing and sending unauthorized information from its host. Such Trojan horses can be classified as spyware as well. Malware embedded by its designer is not limited by this kind of malicious activity. The embedded malware could also be a time bomb [3].

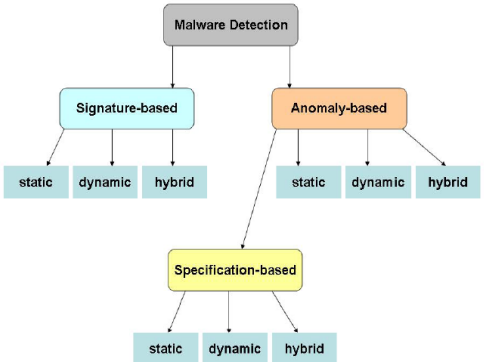

Fig shows hierarchy of malware detection techniques.

For example, malware might create the following scenario; "On May 4, 2010 the compromised system will be programmed to refuse all requests for services." The sheer number and variety of known and unknown malware is part of the reason why detecting malware is a difficult problem. Christodorescu and Jha [4] and McGraw and Morisett [5] provide detailed descriptions of various types of malware. McGraw and Morrisett note that categorizing malicious code has increasingly become more complex as newer versions appear to be combinations of those that belong to existing categories

III. MALWARE DETECTION TECHNIQUES

[2] Techniques used for detecting malware can be categorized broadly into two categories: anomaly-based detection and signature-based detection. An anomaly-based detection technique uses its knowledge of what constitutes normal behavior to decide the maliciousness of a program under inspection. A special type of anomaly-based detection is referred to as specification-based detection. Specification-based techniques leverage some specification or rule set of what is valid behavior in order to decide the maliciousness of a program under inspection. Programs violating the specification are considered anomalous and usually, malicious. Signature-based detection uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection. As one may imagine this characterization or signature of the malicious behavior is the key to a signature-based detection method's effectiveness.

IV. DYNAMIC ANOMALY-BASED DETECTION

[2] In dynamic anomaly-based detection, information gathered from the program's execution is used to detect malicious code. The detection phase monitors the program under inspection during its execution, checking for incons istencies with what was learned during the training phase

Detection Approach Diagram

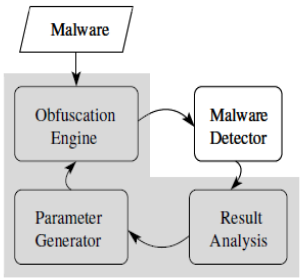

Figure 1: The architecture of the malicious code detector test toolkit.

[6] In this paper we introduce techniques to advance the use of lower-level micro architectural features in the anomaly-based detection of malware exploits. Existing malware detection techniques can be classified along two dimensions: detection approach and the malware features they target, as presented in Figure 1. Detection approaches are traditionally categorized into misuse-based and anomaly based detection. Misuse-based detection flags malware using pre-identified attack signatures or heuristics. It can be highly accurate against known attacks but can be easily evaded with slight modifications that deviate from the signatures. On the other hand, anomaly-based detection characterizes baseline models of normalcy state and identifies attacks based on deviations from these models. Besides known attacks, it can potentially identify novel ones. There are a range of features that can be used for detection: until 2013, they were OS and applicationlevel observables such as system calls and network traffic. Since then, lower-level features closer to hardware such as microarchitectural events have been used for malware detection. Shown in Figure 1, we examine for the first time, the feasibility and limits of anomaly-based malware detection using both architectural and low-level microarchitectural features available from HPCs. Prior misuse-based research that uses microarchitectural features such as [7] focuses on flagging Android malicious apps by detecting payloads. A key distinction between our work and prior work is when the malware is detected. Malware infection typically comprises two stages, exploitation and take-over. In the exploitation stage, an adversary exercises a bug in the victim program to hijack control of the program execution. Exploitation is then followed by more elaborate take-over procedures to run a malicious payload such as a keylogger. Our work focuses on detecting malware during exploitation, as it not only gives more lead time for mitigations but can also act as an early-threat detector to improve the accuracy of subsequent signature-based detection of payloads. The key intuition for the anomaly-based detection of malware exploits stems from the observation that the malware, during exploitation, alters the original program flow to execute peculiar non-native code in the context of the victim program. Such unusual code execution tend to cause perturbations to the dynamic execution characteristics of the program. If these perturbations are observable, they can form the basis of detecting malware exploits.

V. SPECIFICATION AND ANOMALY DETECTION APPROACH TO NETWORK INTRUSION

Although Sekar et al. [8] consider that their detection method as both specification based and anomaly-based, given how we have defined specification-based and anomaly based detection in this report, this method would be more aptly named "A Model-based Approach to Anomaly Detection." Network behavior transitions on events that have arguments, and is modeled with an EFSA, which is an extended finite state automaton. An EFSA can (1) make (2) use a finite set of state variables in which values can be stored. EFSAs model the network interface of the gateway host of the target network.

Sekar et al. leverage statistical properties seen in network traffic to determine the maliciousness of the network events on a target network. For example, the number of timeout transitions that are taken over some subset traces of the EFSA can be taken to identify useful properties about the data stream. Another example would be to analyze the distribution of state variable values seen over some period of time. Anomalous behavior is based on repetition. The authors use the Lincoln Labs 1999 evaluation data. Sekar et al.'s approach were able to detect all attacks in the 1999 data that were within the scope of their method. Their method generated 5.5 false alarms a day, which is low when compared to false alarm rate reported in the Lincoln Labs evaluation data.

VI. STATIC ANOMALY-BASED DETECTION

In static anomaly-based detection, characteristics about the file structure of the program under inspection are used to detect malicious code. A key advantage of static anomaly-based detection is that its use may make it possible to detect malware without having to allow the malware carrying program execute on the host system.During the training phase, a model or set of models are derived that attempt to characterize
the various file types on a syste based on their structural (byte) composition. These models are derived from learning the file types the system intends to handle.The authors' premise is that benign files have predictable regular byte compositions for their respective types. So for instance, benign .pdf files have a unique byte distribution that is different from .exe or .doc files. Any file under inspection that is deemed to vary "too greatly" from the given model or set of models, is marked as suspicious. These suspicious files are marked for further inspection by some other mechanism or decider to determine whether it is actually malicious.

VII. HYBRID ANOMALY-BASED DETECTION

Strider GhostBuster Wang et al. [9] propose a method for detecting a type of malware they refer to as "ghostware." Ghostware is malware that attempts to hide its existence from the Operating System's querying utilities. This is typically done by intercepting the results for these queries and modifying them so traces of the ghostware could not be found/detected via API queries. For example, if a user performs a command to list the files in the current directory, "dir," the ghostware would remove any of its resources from the results returned by the "dir" command.
Wang et al. offer a "cross-view diff-based" approach to detecting these type of malware. In addition to this approach they offer two ways of scanning for the malware, one being an inside-the-box approach and the other an outside-the-box approach. Since there are many layers that return values, and actual arguments must pass through them when a system call is made, many opportunities are afforded to ghostware to intercept function calls. The authors' proposed method to counter this vulnerability will compare the results from a high-level system call like "dir" to a low-level access of the same data without using a system call. An example of a low-level access may be accessing the Master File Table (MFT) directly. This described process is considered the "cross-view diff-based" approach.The inside-the-box approach mandates tht the comparison of the high-level and low-level results are within the same machine.
Hence, an alternative to the inside-the-box approach is the outside-the-box approach. In the outside-the-box approach, another (clean) host performs the low-level access without the target host's knowledge. The high-level scan of the target host is compared to the low-level scan from the clean host. If there is any difference between the lowlevel or high-level scans in either the inside-the-box or outside-the-box approach then ghostware is present in the target host.
In detecting file-hiding ghostware (e.g. ProBot SE and Aphex), the inside-the-box approach did not produce any false positives. However, the outside-the-box approach did produce some false positives for the 10 ghostware the authors used in their experiments. For registry-hiding ghostware (e.g. Hacker Defender 1.0 and Vanquish), virtually no valid false positives were found. The one false positive found, over the six ghostware used in this experiment, was fixed with a minor change. For process/modulehiding ghostware (e.g. Berbew and FU), no false positives were found for the four ghostware used in the experiment. The authors do acknowledge that it is possible for false positives to occur for these type of ghostware, but did not see any during experimentation.

VIII. SELF-NONSELF SPECIFICATION-BASED DETECTION

Specification-based detection is a type of anomaly-based detection that tries to address the typical high false alarm rate associated with most anomaly-based detection techniques. Since specification-based detection is a derivative of anomaly-based detection is also valid for specification-based detection. Instead of attempting to approximate the implementation of an application or system, specification-based detection attempts to approximate the requirements for an application or system. In specification-based detection, the training phase is the attainment of some rule set, which specifies all the valid behavior any program can exhibit for the system being protected or the program under inspection. The main limitation of specification-based detection is that it is often difficult to specify completely and accurately the entire set of valid behaviors a system should exhibit. One can imagine that even for a moderately
complex system, the complete and accurate specification of its valid behaviors can be intractable. Even when it may be straight-forward to express specifications for a system in natural language, it is often times difficult to express this in a form amenable for a machine.

IX. DYNAMIC SPECIFICATION-BASED DETECTION

Approaches classified as dynamic specification-based use behavior observed at runtime to determine the maliciousness of an executable. Monitoring Security-Critical Programs Ko et al. [10] propose a specification-based method for detecting maliciousness in adistributed environment. An implementor would specify the trace policy for the system. A

trace is simply an ordered sequence of execution events, which are essentially the system calls recorded by the auditing mechanism. Ko et al. created a parallel environment
(PE) grammar to address synchronization issues in distributed programs. Audit trails are parsed in real time.
The authors developed an implementation of their approach called Distributed Program Execution Monitor (DPEM). Trace policies were created for 15 Unix programs. One trace policy was created for the rdist program. DPEM was able to catch two violations present in an attack on the rdist program. The violation signal occurred approximately .06 seconds after the actual violation. The authors observed similar delay when they created a trace policy involving passwd and vi. Intrusion attempts on sendmail and binmail were also detected by DPEM in 0.1 seconds.

X. USING DYNAMIC INFORMATION FLOW TO PROTECT APPLICATIONS

Masri et al. [11] describe a tool called the Dynamic Information Flow Analysis (DIFA). This tool was designed specifically for Java applications. This tool has the ability to monitor method calls at runtime by instrumenting the bytecode classes of applications. Hence, information flow can be captured each time a method is invoked by the application and can be compared against known information flow policy. For example, certain directories may be declared, by the policy creator, to be "SensitiveSources," which are objects that may be involved in an illegal information flow. A policy may exist that prohibits certain user/applications from writing this information. This policy would be verified in a "sink" method. The sink methods in this scenario would be any output function invoked by the unauthorized user/applications like write() or send(). This technique can also be used for known malicious information flows, which would then allow this tool to be used as a signature-based tool instead of a specification-based tool.

XI. CONCLUSIONS

By this aim of the malware detection techniques detection will be more improved and we will be able to stop more kinds of malicious behaviors.
As observed there are several anti-viruses developed for protection but still these are not capable enough for providing complete protection.So, there has to be an anti-malware software which protects from all kinds of malware present in network.
In the previous papers the common malicious behavior is observed and formal semantics of them are been detected and the algorithm prepared is for the detection of such obfuscations generally used by hackers.

ACKNOWLEDGMENT

I am very greatful to my guide for all the support and help in my whole research and progress work.

REFERENCES

[1] "An advanced algorithm for malware detection" ,Hazra anjum khan, Kranti kumar dewangan, published in EICSO,IEEE conference proceeding 2015.
[2] "Survey on malware detection technique"Aditya P.Mathure 2005.
[3] c.landwehr,a.bull,j.Mc dermott.ataxonomy of computer program security flaws,(csur),26(3):211-254,1994.
[4] M.CHRISTODORESCU and S.JHA.testing malware detection.in july 2004
[5] Unsupervised anomaly-based malware detection using hardware features.adrian tang, simha sethumadhavan and salvatore stolfo, new york,usa,2014
[6] Demme, J., Maycock, M., Schmitz, J., Tang, A.,Waksman, A., Sethumadhavan, S.,Stolfo, S.: On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture pp 559 ISCA 2013.
[7] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: A new approach for detectin network intrusions. ACM Computer and Communication Security Conference, 2002.
[8] Y. M. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski. Detecting stealth software with strider ghostbuster. In Proceedings of the 2005 International Conference.

