ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3. Issue 10 October, 2014 Page No. 8699-8703

A Pragmatic Study of Malwares to Enrich Application for Self Defence

Sanket S.Deshpande [1], Prachetus H.Dindore [2], Shubham N.Munot [3], Prabhat kumar Prabhakar [4], Prof.Anuradha S.Deokar [5]

[1]Department of Computer Engineering
A.I.S.S.M.S College of Engineering
Pune,India
dsanket.93@rediffmail.com

[2] Department of Computer Engineering
A.I.S.S.M.S College of Engineering
Pune,India
prachetusdindore@gmail.com

[3]Department of Computer Engineering
A.I.S.S.M.S College of Engineering
Pune,India
Shubhammunot27@gmail.com

[4]Department of Computer Engineering
A.I.S.S.M.S College of Engineering
Pune,India
pprabhakar003@gmail.com

[5]Department of Computer Engineering
A.I.S.S.M.S College of Engineering
Pune,India
deokar.anu@gmail.com

Abstract: ***In the era of internet the probability of system or application being vulnerable to malwares such as viruses, Trojans, botnets etc. has provoked data corruption, data manipulation, security breaching and so on. Different techniques like antivirus and firewalls have emerged to combat against malware attacks. However existing signature based detections are unable to counteract anomalous behaviour of specific applications. There exist various behaviour based techniques which detects malicious content by observing applications at run-time. The focal point of this paper is on immunising an application against specific threats.***

Keywords: malwares, signature, anomaly, system calls, automaton.

1. Introduction

Certain programs are designed with the intent of disrupting system operations, corrupting databases, illegal monitoring of user data etc. Such programs are collectively termed as Malwares. They were initially used for the purpose of experiments or pranks. But in today's world, they are generated to steal, monitor or destroy personal, financial or business information. There exists various types of malwares which include viruses, worms, Trojan, botnets, root kits, key loggers etc which can break through any security defences

Current malware detection techniques mainly consist of AV-Scanners etc. They are mainly based on Signature-based detection. Signature-based revealing techniques mainly use extracted byte sequence i.e. signature of suspicious instructions and data. These signatures are used to detect malwares by matching them with signatures extracted from target machine. An attack of a malware, whose signature doesn't exist in the database of the detection tool, hence can go unnoticed. There exist various polymorphism and obfuscation techniques which make malwares unrecognizable and hence used to bypass the AV-Scanners.

There are basically two types of malware detection techniques – Signature based detection and Anomaly based Detection. Basic methodology of Signature based detection is discussed in short above. Also there exists polymorphism based techniques which deobfuscates malwares before providing them to malware detectors. An anomaly-based detection technique uses its knowledge of what differs normal behaviour from malicious behaviour of a program under inspection. Many solutions based on anomaly detection technique use machine learning algorithms but don't take into account semantics of subroutine call sequences. Some obfuscation techniques may evade detection using those flaws.

A. Basic Terminologies:

1) MALWARE:

"Malware" is short for malicious software and used as a single term to denote spy wares, worms, viruses, etc. Malware is intended to cause damage to a standalone computer or a networked pc.

2) SYSTEM CALL:

A system call is how a program requests a service from an operating system's kernel. System calls provide an essential interface between a process and the operating system.

3) POLYMORPHISM:

Polymorphism is the feature of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

4) SIGNATURE:

In order to describe the behaviour of computer program formally --- the verification of the two components syntax and semantics are essential. The signature determines the syntactical components. It provides the available data formats (i.e. sorts) and the available operations defined on them.

5) OBFUSCATION:

Obfuscation (or beclouding) is the hiding of intended meaning in communication, making communication confusing, wilfully ambiguous, and harder to interpret.

6) FINITE STATE AUTOMATA:

It is conceived as an abstract machine that can be in one of a number of finite states. At a given time the machine is in only one state. The next state and output of an FSM is a function of the input and of the current state.

7) DYNAMIC BINARY INSTRUMENTATION:

Dynamic Binary Instrumentation (DBI) provides the technique for analysing the behaviour of a binary application. It is analyzed at run time by the addition of instrumentation code. After being added, this instrumentation code runs as piece of the normal instruction stream.

Instrumentation refers to an ability to monitor or measure the level of performance of a product to write trace information and to identify errors. Programmers device instrumentation in the form of code instructions that monitor specific components in a system (for instance, the instructions may possibly output logging information to appear on screen).

2. LITERATURE SURVEY

Our main focus is on systems based on behavioral malware detection with special emphasis on immunizing applications against specific malwares. We have taken into account the following eight methodologies for covering various aspects of malware detection.

A. First Approach

Any privileged process has root access and thus use system calls and hence have maximum chances of damaging system, if attacked by malware. Authors [1] — in 1996 developed an artificial immune system which detects any abnormal behaviour of processes by observing sequence of system calls hence limit damage to system.

Authors have given information about an approach of Fink, Levitt and Ko, which focuses on determining normal behaviour for privileged processes running as root. Forrest et al had previously prepared a system similar to theirs. But that system was at file-authentication level.

However this system fails sometimes while partial or approximates matching and system cannot detect certain attacks.

B. Second Approach

Authors [2] — describes the detection of intrusion which is in terms of system calls.

They proposed i.e. introduced a simple IDS based on monitoring system calls by active and privileged process. Also comparison of different data modelling methods is done.

Computational efficiency is high due to simple approach of distinguishing normal and intrusive behaviour by traces of system calls.

C. Third Approach

There are many techniques using learning program behaviours, but FSA(Finite-State Automaton) based techniques seems effective, but in previous attempts FSA – learning was computationally expensive and not much automated. Authors [3] — developed a technique of FSA-learning which is fully automatic and efficient and consume less time and space.

System uses program counter, which is stored when a certain privileged process turned to kernel mode using system call, thus it can be used to trace system call and its order in a procedure stack. Each distinct value of program counter is used as a state in a FSA. Transitions between various states can be shown with the help of pair of current and previous system calls.

This system successfully captures DoS attacks, Trojans, buffer overflows but fails to capture certain attacks which use system call argument values.

D. Fourth Approach

Authors [4] — describes a malware detection algorithm that includes instruction semantics to detect malicious program behaviour.

They propose a tool which is based on pattern matching tool. This tool first disassembles binary program into blocks and then generates a control flow graph(CFG).At run-time the tool matches each block with templates(malicious instruction sequence).If match is found then it is discarded else it continues to run.

It detects variants of malwares with relatively low run-time overhead. It is also resistant to common obfuscation used by hackers.

E. Fifth Approach

Authors [5] — describes two advanced methods to handle buffer overflow. First method intercepts calls to library functions known to be vulnerable. Second method uses binary modification of process memory to verify critical elements before use.

These methods can transparently protect processes against stack smashing attacks by corrupting return addresses.

F. Sixth Approach

Authors [6] — describes an alternative to the signature based approach i.e. behavioural detection. In this approach run-time behaviour of an application is monitored and compared against malicious and/or normal profiles.

A behavioural classifier is trained by the normal behaviour of typical services as well as malicious patterns for currently known mobile malwares. The classifier with behaviour signature database is deployed or installed to handsets. The monitor agent assembles the application behaviour in the form of API calls/events and given report to detection agent. It then performs machine learning classification with preloaded classifier that a program is innocent or malicious. Since encryption/decryption doesn't alter application behaviour, multiple malware variants generated at runtime can be detected with a single behaviour specification, hence resilient to polymorphic worms.

Malware writers make use of different polymorphism and obfuscation techniques in order to avoid detection by normal malware detectors. Authors[7]— describe the use of a malware transformer which is able to reverse the obfuscated versions done by the malware writers.

G. Seventh Approach

A malware detector is one which is able to recognize and notice malwares before their entry into the system. System proposed by [7] — includes a malware transformer which takes an obfuscated application as its input and generates obfuscation free application as its output. The transformer is able to handle three kinds of obfuscations like obfuscation done by reordering of code, obfuscation through junk insertion and packing obfuscation. For each of the three obfuscation technique a corresponding transformer algorithm has been applied to reverse the effect of obfuscation.

The addition of malware transformer as a part of the detector leads to an effective software and its maintenance along with the transformation performed. However the set of obfuscations handled by the transformer remain limited.

H. Eighth Approach

Authors [8]— have used a different approach. In this approach, system divides executable into blocks which describes data flows in terms of regular expressions and data invariants.

Then they have generated execution trace with the help of DBI (Dynamic Binary Instrumentation). Execution trace is converted into regular expressions.

They enforced security policies and checked these at run-time. They have created a model and deploy it with executable to compare executable behavior with security policies. This approach is shown in detail in Fig.1

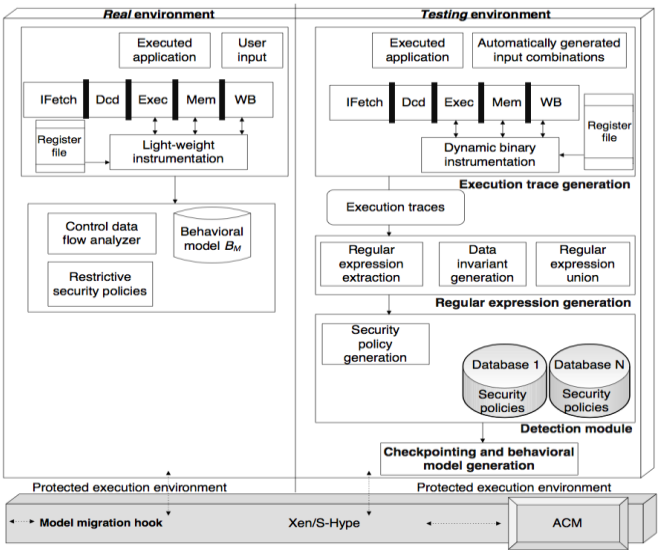

Figure 1: System Framework

3. DISCUSSION

First two approaches have used system calls and its variants to prevent privileged processes, thus securing system from damage. The third approach has used FSA to automate the detection of malware. The fourth approach has used CFG to detect malicious behaviour. Fifth approach uses library function calls and process memory. The sixth approach has used machine learning techniques to identify malicious data

in mobile handsets .In seventh approach system contains a deobfuscator which helps conventional malware detectors. In eight approaches a model of specific security policies is developed which is compared with executables at run time to detect malicious behaviours.

However system calls approach remains restricted to processes which have root access. So first three approaches are not applications specific. Considering machine learning techniques, the mainly depend on how well are they trained. Malware deobfuscators are restricted by the limited set of obfuscations and are susceptible to Zero Day attacks.

The fourth and eighth approaches have chosen a path of monitoring the application at run time. And if at run time behaviour of the executable deviates from the model then security policies are enforced. We thus look forward to extend these two approaches by bringing sophistication in the input sequence generation and run time monitoring.

4. CONCLUSION

The various techniques discussed above in this paper thus satisfy the requirements of enriching the applications in their own characteristic way. Among various methodologies that have been used, one striking feature which is in common is observing behaviour of an application at run time with an intention of detecting malicious data. Among all the approaches last approach has better future due to less power consumption, better detection rates and being application specific.

REFERENCES

[1] Hofmeyr S., Forrest S., Somayaji T., Longstaff T.: A sense of self for Unix processes. : IEEE convention on Security - Privacy, (1996).

[2] Warrender C., Pearlmutter B., Forrest S.: Detecting intrusions using system calls: Alternative data models. In: Proceedings of IEEE convention on Security - Privacy, (1999).

[3] Bendre M., Dhurjati D., Sekar R., Bollineni P.:A fast automaton based method for detecting anomalous program behaviors. In: IEEE convention on Security - Privacy, (2001).

[4] Jha S., Christodorescu M., Seshia S., Song D., Bryant R.: Semantics-aware malware detection. In: Proceedings of IEEE convention on Security - Privacy, (2005).

[5] Singh N., Baratloo A., Tsai T.: Transparent run-time defense against stack smashing attacks. :USENIX Annual Technical convention, (2000).

[6] Shin K.G., Bose A., Hu X., Park T.: Behavioral detection of malware on mobile handsets. In: Proceedings of International Conference on Mobile s, applications, andServices, (2008).

[7] Kinder J.,Christodorescu M., Jha S.,Katzenbeisser S., Veith H.: Software transformations to improve malware detection. J Comput Virol 3 (2007).

[8] Aaraj N., Raghunathan A., Jha N.K.: Virtualization-based framework for malware defense. In: Proceedings of Convention Detection of Intrusions and Malware and Vulnerability, Assessment (2008).

AUTHOR PROFILE

Sanket S. Deshpande is pursuing Bachelor's Degree in Computer Engineering in Savitribai Phule Pune University from A.I.S.S.M.S College Of Engineering Pune,Maharashtra,India.

Prachetus H. Dindore is pursuing Bachelor's Degree in Computer Engineering in Savitribai Phule Pune University from A.I.S.S.M.S College Of Engineering Pune,Maharashtra,India.

Shubham N.Munot is pursuing Bachelor's Degree in Computer Engineering in Savitribai Phule Pune University from A.I.S.S.M.S College Of Engineering Pune,Maharashtra,India.

Prabhat kumar Prabhakar is pursuing Bachelor's Degree in Computer Engineering in Savitribai Phule Pune University

from A.I.S.S.M.S College Of Engineering
Pune,Maharashtra,India.

Prof.Anuradha S.Deokar is Assistant Professor in Department Of Computer Engineering in Savitribai Phule Pune University from A.I.S.S.M.S College Of Engineering Pune,Maharashtra,India.