ISSN 2348 – 9928
IJAICT Volume 3, Issue 3, March 2016 Doi:01.0401/ijaict.2014.05.07 Published on 05 (3) 2016

FAULT PREDICTION IN OBJECT ORIENTED SYSTEMS USING MALICIOUS CODE DETECTION

Mr. N. Rajkumar
Assistant professor,
Department of computer science,
SVS College of Engineering,
Coimbatore, Tamilnadu, India

Ms. C. Viji
Assistant professor,
Department of computer science,
SVS College of Engineering,
Coimbatore, Tamilnadu, India

Dr. S. Duraisamy
Professor & Head,
Department of Computer Applications,
Sri Krishna College of Engg & Tech,
Coimbatore, Tamilnadu, India

Abstract— There are many classification algorithms were employed successfully for the detection of unknown malicious code. Most of these studies extracted features based on byte n-gram patterns in order to represent the inspected files. In this study were present the inspected files using OpCode n-gram patterns which are extracted from the files after disassembly. The OpCode n-gram patterns are used as features for the classification process. The classification process main goal is to detect unknown malware within a set of suspected files which will later be included in antivirus software as signatures. A rigorous evaluation was performed using a test collection comprising of more than 30,000 files, in which various settings of OpCode n- gram patterns of various size representations and eight types of classifiers were evaluated. A typical problem of this domain is the imbalance problem in which the distribution of the classes in real life varies. We investigated the imbalance problem, referring to several real-life scenarios in which malicious files are expected to be about 10% of the total inspected files. Lastly, we present a chronological evaluation in which the frequent need for updating the training set was evaluated. Evaluation results indicate that the evaluated methodology achieves a level of accuracy higher than 96% (with TPR above 0.95 and FPR approximately 0.1), which slightly improves the results in previous byte n-gram representation.

Keywords— N-Grampattern, Opcode, Code Emulation, Pattern Based Scanning.

I. INTRODUCTION

Modern computer and communication infrastructures are highly susceptible to various types of attacks. A common method of launching these attacks is by means of malicious software (malware) such as worms, viruses, and Trojan horses, which, when spread, can cause severe damage to private users, commercial companies and governments. The recent growth in high-speed Internet connections enable malware to propagate and infect hosts very quickly, therefore it is essential to detect and eliminate new (unknown) malware in a prompt manner [1].

Anti-virus vendors are facing huge quantities (thousands) of suspicious files every day [2]. These files are collected from various sources including dedicated honey pots, third party providers and files reported by customers either automatically or explicitly. The large amount of files makes efficient and effective inspection of files particularly challenging. Our main goal in this study is to be able to filter out unknown malicious files from the files arriving to an anti-virus vendor every day. For that, we investigate the approach of representing malicious files by OpCode expressions as features in the classification task.

Several analysis techniques for detecting malware, which commonly distinguished between dynamic and static, have been proposed. In dynamic analysis (also known as behavioral analysis) the detection of malware consists of information that is collected from the operating system at runtime (i.e., during the execution of the program) such as system calls, network access and files and memory modifications. This approach has several disadvantages [3]. First, it is difficult to simulate the appropriate conditions in which the malicious functions of the program, such as the vulnerable application that the malware exploits, will be activated. Secondly, it is not clear what is the required period of time needed to observe the appearance of the malicious activity for each malware [4].

In static analysis, information about the program or its expected behavior consists of explicit and implicit observations in its binary/source code. The main advantage of static analysis is that it is able to detect a file without actually executing it and thereby providing rapid classification.

Static analysis solutions are primarily implemented using the signature-based method which relies on the identification of unique strings in the binary code. While being very precise, signature-based methods are useless against unknown malicious code .Thus, generalization of the detection methods is crucial in order to be able to detect unknown malware before its execution. Recently, classification algorithms were employed to automate and extend the idea of heuristic-based methods. In these methods the binary code of a file is represented, for example, using byte sequence (i.e., byte n-grams), and classifiers are used to learn patterns in the code in order to classify new (unknown) files as malicious or benign. Recent studies, which we survey in the next section, have shown that by using byte n-grams to represent the binary file

features, classifiers with very accurate classification results can be trained, yet there still remains room for improvement.

Another aspect in the maintenance of such a framework is the importance of updating the training set with new known malicious files. This is intuitively important, because the purpose of malicious files changes over time and accordingly the patterns within the code. Moreover, these malicious files are written in varying frameworks which result in differing patterns. However, it is not clear to what extent it is essential to retrain the classifier with the new files. For this purpose we designed a chronological experiment, based on a dataset including files from the years 2000 to 2007, and trained each time on files until year k and tested on the following years [5].

II. EXISTING SYSTEM

There are many different technologies available to detect viruses most of which rely on the internal structure (rather than the behavior) of the virus. Although the behavior of each of the permutations of a metamorphic virus is the same, the structure is different which means they can become difficult to detect depending on the amount of variation [6]. There are some detection methods which detect suspicious ability or behavior within a program, such as heuristic analysis; however these methods are rarely used as a sole means of virus protection as they are normally prone to false-positives. The following subsections will discuss various existing methods

2.1 Code Emulation

Code emulation was briefly mentioned when discussing encrypted and polymorphic viruses as a possible means of retrieving the unencrypted form of the virus body. Using code emulation can be an effective tool when detecting viruses, since it involves emulating software on a virtual machine rather than a real processor. Because the virus is in a controlled environment, the system emulating the virus will not run any risk of harmful side-effects of the virus [7].

2.2 Pattern based scanning

One of the most common approaches to detecting viruses is called string or signature scanning. At a high level, signature scanning involves a set of signatures, or a string of bytes which can contain wild cards, which are found in viruses but not found in non-malicious code. These signatures often contain non-contiguous code, using wild cards where differences lie. These wild cards allow the scanner to detect if virus code is padded with other junk code [8]. This kind of detection requires research on known viruses, and patterns within each virus need to be studied so that these signatures can be found [9].

Although signature scanning works well on most viruses, a metamorphic virus could potentially create enough variation within the application making it nearly impossible to create a reliable signature [10].

III. PROPOSED SYSTEM

The approach which is proposed in this paper will be focused on the detection of unknown malware based on its binary code content. The authors of were the first to introduce the idea of applying Machine Learning (ML) methods for the detection of different malwares based on their respective binary codes. Three different feature extraction (FE) approaches were employed: features extracted from the Portable Executable (PE) section, meaningful plain-text strings that are encoded in programs files, and byte sequence features [11].

Detecting Unknown Malware using Byte N-Grams Patterns
Over the past decade, several studies have focused on the detection of unknown malware based on its binary code content. The authors of [16] were the first to introduce the idea of applying Machine Learning (ML) methods for the detection of different malwares based on their respective binary codes. Three different feature extraction (FE) approaches were employed:

Features extracted from the Portable Executable (PE) section, meaningful plain-text strings that are encoded in programs files, and byte sequence features [12].

Representing Executables using OpCodes [13] An OpCode (short for operational code) is the portion of a machine language instruction that specifies the operation to be performed. A complete machine language instruction contains an OpCode and, optionally, the specification of one or more operands. The operations of an OpCode may include arithmetic, data manipulation, logical operations, and program control.

The OpCodes, being the building blocks of machine language, have been used for statically analyzing application behavior and detecting malware [14]. addressed the tracking of malware evolution based on OpCode sequences and permutations. Data mining methods (Logistic Regression, Artificial Neural Networks and Decision Trees) are used in to automatically identify critical instruction sequences that can distinguish between malicious and benign programs. The evaluation showed a high accuracy level of 98.4%. Bilar examines the difference of statistical OpCode frequency distribution in malicious and non-malicious code. A total of 67 malware executables were compared with the aggregate statistics of 20 non-malicious samples. The results show that malicious soft-

ware OpCode distributions differ significantly from non-malicious software and suggest that the method can be used to detect malicious code. The approach in presents a single case in our methodology; in this paper we test several OpCode n-gram sizes while Bilar used only 1-gram. Based on our experiments, using OpCode sequences improves the detection performance significantly. Santos et al. Used the OpCode n-grams (of size n=1, 2) representation to ascribe malware instances to their families by measuring the similarity between files. This is, however, different from our goal in which we attempt to classify unknown suspicious files as malicious or benign in order to detect new malware [15].

Our approach also stems from the idea that there are families of malware such that two members of the same family share a common "engine." Moreover, there are malware generation utilities which use a common engine to create new malware instances; this engine may even be used to polymorph the threat as it propagates. When searching for such common engines among known malware, one must be aware that malware designers will attempt to hide such engines using a broad range of techniques. For example, these common engines may be located in varying locations inside the executables, and thus may be mapped to different addresses in memory or even perturbed slightly. To overcome such practices, we suggest disregarding any parameters of the OpCodes. We believe that disregarding the parameters would provide a more general representation of the files, which is expected to be more effective for purposes of classification into benign and malicious files.

3.1 First Method. Dataset Creation

We created a dataset of malicious and benign executables for the Windows operating system, the system most commonly used and attacked today. This malicious and benign file collection was previously used in. We acquired 7,688 malicious files from the VX Heaven website. To identify the files, we used the Kaspersky antivirus. Benign files, including executable and DLL (Dynamic Linked Library) files, were gathered from machines running the Windows XP operating system on our campus.

The benign set contained 22,735 files. The Kaspersky anti-virus program was used to verify that these files did not contain any malicious code.

Some of the files in our collection were either compressed or packed. These files could not be disassembled by disassembler software and therefore, after converting the files into OpCode representation we ended up with 5,677 malicious and 20,416 benign files (total of 26,093 files).

Code obfuscation is a prominent technique used by hackers in order to avoid detection by security mechanisms (e.g., anti-viruses and intrusion detection systems).These techniques are also applied on benign software for copyrights protection purposes. Packing and compressing files can be achieved by using off-the-shelf packers such as Armadillo, UPX and Themida. In such cases, static analysis methods might fail to correctly classify a packed malware. Several solutions to the challenge of packed code were suggested (e.g., Ether, McBoost, PolyUnpack). These methods were proposed for automatic unpacking of packed files by applying either static or dynamic analysis. Evaluation performed in these studies showed that unpacking files before being classified increase the classification accuracy. Our proposed method can use such an approach in order to overcome packed files. In addition, we would like to point out that classifying benign files is also useful and can reduce the load of inspecting suspicious (or unknown) files. Also, the large number of malware files in our dataset that could be dissembled indicates that in order to appear benign and to pass security mechanisms (that are configured to block content that is encrypted\obfuscated and cannot be inspected), these techniques are not always used by hackers.

3.2 Second Method. Data Preparation and Feature Selection

To classify the files we had to convert them into a vectorial representation. We had two representations, the known one, often called byte n-grams, which consists of byte sequences of characters extracted from the binary code, and the second OpCode n-grams represented by sequences of OpCodes. Using disassembler software, we extracted a sequence of OpCodes from each file representing execution flow of machineextracted from the binary code, and the second OpCode n-grams represented by sequences of OpCodes. Using disassembler software, we extracted a sequence of OpCodes from each file representing execution flow of machine operations. Subsequently, several OpCode n-gram lengths were considered where each n-gram was composed of n sequential OpCodes. This process is presented in Figure 1.

The process of streamlining an executable starts with disassembling it. The disassembly process consists of translating the machine code instructions stored in the executable to a more human-readable language, namely, Assembly language. The next and final step in streamlining the executable is achieved by extracting the sequence of OpCodes generated during the disassembly process. The extracting of sequences is in the same logical order in which the OpCodes appear in the executable, disregarding the extra information available (e.g., memory location, registers, etc.)The size of vocabularies (number of distinct n-grams)

extracted for the OpCode n-grams representation were of 515, 39,011, 443,730, 1,769,641, 5,033,722 and 11,948,491, for 1-gram, 2-gram, 3-gram, 4-gram, 5-gram and 6-gram, respectively.

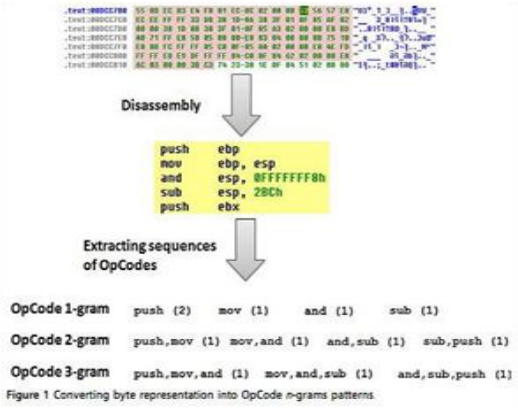

Figure 1 Converting byte representation into OpCode n-grams patterns

Fig 1 : File processing flow diagram

Later, the normalized term frequency (TF) and TF inverse document frequency (TFIDF) representations were calculated for each OpCode n-grams patterns in each file. The TF and TFIDF are well known measures in the text categorization field. In our domain, each n-gram is analogous to a word (or a term) in a text document. The normalized TF is calculated by dividing the frequency of the term in the document by the frequency of the most frequent term in a document. The TFIDF combines the frequency of a term in the document (TF) and its frequency in the whole document collection, denoted by document frequency (DF). The term's (normalized) TF value is multiplied by the IDF = log (N/DF), where N is the number of documents in the entire file collection and DF is the number of files in which it appears.

Input Design

The first step in system design is to design the input and output within predefined guidelines. In input design, user originated inputs are converted into computer based format. In output design, the emphasis is on producing the hard copy of the information requested of displaying the output on CRT screen in a predefined format.

Inaccurate input data are the most common causes of errors in data processing. Project message and appropriate sections can control errors committed by data entry operators. The following features have been incorporated into the input design of the proposed system.

3.2.1 Easy data input

Data entry screens have been designed in a manner much similar to old systems. Each form has controls for insertion, updating and exit. Appropriate messages are provided in the message area, which prompts the user in entering the right data. Erroneous data inputs are checked while user inputs.

3.2.2 Use Friendliness

User is never left in a state of confusion as to what is happening; instead appropriate error and acknowledgement messages are sent. Error capturing is used to indicate the error codes and specific error messages.

3.2.3 Consistent Format

A fixed format is adopted for, displaying the title and messages. Every screen has buttons, which displays the operation that can be performed after data entry. They are normally done at the touch of a key or mouse.

3.2.4 Interactive dialogue

The system engages the user in an interactive dialogue. The system is able to extract missing or omitted information from the user by directing the user through appropriate messages, which are displayed.

Input design is the process of converting user oriented inputs to computer based format. It also includes determining the record media, method of input, speed of capture, and entry into the system.

Consideration can be given to

a) Type of input
b) Flexibility of format
c) Speed
d) Accuracy
e) Verification methods
f) Ease of correction
g) Need for specialized documentation
h) Storage and handling requirements
i) Automatic features
j) Hard copy requirements Security
k) Environments of data capture
l) Portability
m) Compatibility with other system
n) Cost etc.

Keyboard may be used as in input media. The data are displayed on cathode ray tube screen for verification. Inaccurate input data are the most common cause of errors in data processing. Errors entered by the user can be controlled by input design.

3.2.5 Output Design

During the output design phase the necessary outputs are planned and designed based on user the objective of the reports is to present the data in the way that the management can present the data quickly and clearly. Each and every page

is designed in such a way they can be taken printouts with required information. This information helps the user to provide a neat and clear presentation about the student information.

Whether the output is formatted report or a simple listing of the contents of a file, a computer process will produce the output.

- A Document
- A Message
- Retrieval from a data store
- Transmission from a process or system activity
- Directly from an output sources

The Output will be object detection with separate part detection.

IV. CONCLUSION

In this work we define a new malicious code detection algorithm. That will reduce the unknown code in the developed program that will improve the quality of software, improves reusability and easy debugging. In this work the information are extracted into opcode n-gram representation that will help to identify the unknown code. The output of this file helps to predict the fault in the object oriented systems.

REFERENCES

[1] Amos. Dange, Prof. Dr. S. D. Joshi, "Fault Prediction in Object Oriented System Using the Coupling and Cohesion of Classes", IJCSMS International Journal of Computer Science and Management Studies, Vol.11, Issue 02, pp. 48-51, Aug 2011.

[2] Andrian Marcus, Denys Poshyvany k," Using the Conceptual Cohesion of Classes for Fault Prediction inan Object-Oriented Systems,"IEEE Transactionson Software Engineering, VOL.34, NO.2, pp 287-300, march /april2008.

[3] Arisholm.E., Briand,L. C., and Foyen.A, "Dynamic coupling measurement for OO software",IEEETSE,vol.30,no.8,pp. 491-506,2004

[4] Bansiya,J.and Davis,C. G., "A hierarchical model for object-Orienteddesign quality assessment",IEEETSE vol.28,no.1 ,pp.4-17.,2002

[5] Briand.l, Devanbu.p and Mello.w,"An investigation into coupling measures for C++," proceedings of ICSE, 1997.

[6] Briand, S.Morasca, and V.R.Basili, "Property-Based Software Engineering Measurements," IEEETrans.SoftwareEng., vol.22, no.1, pp.68-85, Jan.1996.

[7] Briand,L.C., Wüst,J.,Daly,J.W., and Porter, V.D ,"Exploring the relationship between design measures and software quality in object-oriented systems", Journal of System and Software,vol.51,no.3,pp.245-273.,2000.

[8] Chidamber, S.R. and Kemerer,C.F., "A Metrics Suite for Object Oriented Design ",in IEEE Transaction In Software Engineering, Vol 20,No 6,june 1994.

[9] Deepak Arora , Pooja Khanna and AlpikaTripathi,Shrpra Sharma and SanchikaShukla," Software quality estimation through object oriented design metrics," In International Journal Of Computer Science And Network Security,Vol 11, No 4,April 2011.

[10] Guigui,Paul D. Scott , "Measuring software component reusability by coupling and cohesion," In journal of computers,vol.4,No 9,September 2009.

[11] Kavitha, Dr.A.Shanmugam, "Dynamic coupling of object oriented software using trace events,"In IEEE Transaction,2008.

[12] Lawrie,D.,Feild,H., and Binkley,D.,"Leveraged Quality Assessment using Information Retrieval Techniques", in ICPC'06, pp.149-158.,2006

[13] Mishra, B, ShuklaK. , "Impact of attribute selection ondefect proneness prediction in OO software,"2nd International Conference on Computer and Communication Technology (ICCCT), IEEE Conference Publications, PP: 367-372, 2011.

[14] Subramanyam and Krishnan, "Empirical analysis of CK metrics for OOD complexity:Implication for software defect," IEEE transaction on software engineering,2003.

[15] SukainahHusein and Alan Oxley "A coupling and cohesion metrics suite for object oriented software", international conference on computer technology and development,IEEE conference publication,2009.

[16] Újházi B , Ferenc R, Poshyvanyk DGyimothy.T, "New Conceptual Coupling and Cohesion Metrics for Object-Oriented Systems,"10th IEEE working Conference, Pp. 33 - 42 ,2010.