Detection and Identification of new malware in Network Security

C. Rajagopal
Department of Information Technology
Saveetha School Of Engineering
chennai

Abstract - Spyware represents a serious threat to confidentiality relays the information across the Internet to a third party since it may result in loss of control over private data for location this type of software might collect the data and send it to the third party of the users without the user consent. Programs that have the potential violate the privacy and security of the system can be labeled as privacy software. These are includes the spyware, adware, trojans, greyware and backdoors. It may compromise the confidentiality, integrity and availability of the system and may obtain sensitive information without user consent this is useful for the marketing company and also generates the income for the from online ads distribution through adware. Originally, viruses represented the only major malicious threats to computer users and since then much research has been carried out in order to successfully detect and remove viruses from computer systems. Vendors are embedded spyware in which is installed with the application or by using the hacking methods the installed spyware may be capable of capturing keystrokes ,taking screen shorts, saving authentication credentials, storing personal email address and web form data and thus may obtain behavioral, personal information about users.

Index Terms— Computer security, malware, control flow, structural classification, structured control flow, unpacking.

I. INTRODUCTION

Malware, short for malicious software, means a variety of forms of hostile, intrusive, or annoying software or program code. Malware is a pervasive problem in distributed computer and network systems. According to the symantec internet threat report [1], 499,811 new malware samples were received in the second half of 2007. F-secure additionally reported, "as much malware [was] produced in 2007 as in the previous 20 years altogether"[2]. Detection of malware is important to a secure distributed computing environment.

The predominant technique used in commercial anti-malware systems to detect an instance of malware is through the use of malware signatures. Malware signatures attempt to capture invariant characteristics or pat-terns in the malware that uniquely identifies it. The patterns used to construct a signature have traditionally derived from strings of the malware's machine code and raw file contents [3, 4]. String based signatures have remained popular in commercial systems due to their high efficiency, but can be ineffective in detecting malware variants.

Malware variants often have distinct byte level representations while in principal belong to the same family of malware. The byte level content is different because small changes to the malware source code can result in significantly different compiled object code. In this paper we describe malware variants with the umbrella term of polymorphism. Polymorphism describes related malware sharing a common history of code. Code sharing among variants can be derived from autonomously self mutating malware, or manually copied by the malware creator to reuse previously authored code.

A. Existing Approaches and Motivation

Static analysis incorporating n -grams [5, 6], edit distances [7], api call sequences [8], and control flow [9-11] have been proposed to detect malware and their polymorphic variants. However, they are either ineffective or inefficient in classifying packed and polymorphic malware.

A malware's control flow information provides a char-acteristic that is identifiable across strains of malware variants. Approximate matchings of flowgraph based characteristics can be used in order to identify a greater number

of malware variants. Detection of variants is possible even when more significant changes to the malware source code are introduced.

Control flow has proven effective [9, 11, 12], and fast algorithms have been proposed to identify exact isomorphic whole program control flow graphs [13] and related information, yet approximate matching of program structure has shown to be expensive in runtime costs. Poor performance in execution speed has resulted in the absence of approximate matching in endhost malware detection.

To hinder the static analysis necessary for control flow analysis, the malware's real content is frequently hidden using a code transformation known as packing. Packing is not solely used by malware. Packing is also used in software protection schemes and file compression for legitimate software, yet the majority of malware also uses the code packing transformation. In one month during 2007, 79% of identified malware was packed. Additionally, almost 50% of new malware in 2006 were re-packed versions of existing malware .

Unpacking is a necessary component to perform static analysis and to reveal the hidden characteristics of mal- ware. In the problem scope of unpacking, it can be seen that many instances of malware utilize identical or similar packers. Many of these packers are also public, and malware often employs the use of these public packers. Many instances of malware also employ modified versions of ware in any of these scenarios, in addition to unpacking novel samples, provides benefit in revealing the malware's real end a necessary component for static analysis and accurate classification.

Automated unpacking relies on typical behavior seen in the majority of packed malware hidden code is dynamically generated and then executed. The hidden code is naturally revealed in the process image during normal execution. Monitoring execution for the dynamic generation and execution of the malware's hidden code can be achieved through emulation. Emulation provides a safe and isolated environment for malware analysis.

Malware detection has been investigated extensively, however shortcomings still exist. For modern malware classification approaches, a system must be developed that is not only effective against polymorphic and packed malware, but that is also efficient. Unless efficient systems are developed, commercial antivirus will be unable to implement the solutions developed by researchers. We believe combining effectiveness with real time efficiency is an area of research which has been largely ignored. For example, the malware classification investigated in [5, 6, 9-11] has no analysis or evaluation of system efficiency. We address that issue with our implementation and evaluation of malwise.

In this paper we present an effective and efficient system that employs dynamic and static analysis to automatically unpack and classify a malware instance as a variant, based on similarities of control flow graphs.

B. Contributions

This paper makes the following contributions. First, we propose using string based signatures to represent control flow graphs and a fast classification algorithm based on set similarity to identify related programs in a database. Second, we propose using two algorithms to generate string signatures for exact and approximate identification of flow graphs. Exact matching uses a graph invariant as a string to heuristically identify isomorphic flow graphs. Unpacking using application level emulation that is equally capable of desktop antivirus integration. The automated unpacker is capable of unpacking known samples and is also capable of unpacking unknown samples. We also propose an algorithm for determining when to stop emulation during unpacking using entropy analysis. Finally, we implement and evaluate our ideas in a novel prototype system called malwise that performs automated unpacking and malware classification.

C. Structure of the Paper

The structure of this paper is as follows: section 2 describes related work in automated unpacking and malware classification; section 3 refines the problem definition and our approach to the proposed malware classification system; section 4 describes the design and implementation of our prototype malwise system; section 5 evaluates malwise using real and synthetic malware samples; finally, section 8 summarizes and concludes the paper.

II. RELATED WORKS

A. Automated Unpacking

Automated unpacking employing whole system emulation was proposed in renovo and Pandora's bochs. Whole system emulation has been demonstrated to provide effective results against unknown malware samples, yet is not completely resistant to novel attacks. Renovo and Pandora's bochs both detect execution dynamically generated code to determine when unpacking is complete and the hidden code is revealed .an alter- native algorithm for detecting when unpacking is complete was proposed using execution histograms in hump-and-dump . The hump-and-dump was proposed as potentially desirable for integration into an emulator. Poly unpack proposed a combination of static and dynamic analysis to dynamically detect code at runtime which cannot be identified during an initial static analysis. The main distinction separating our work from previously proposed automated unpackers is our use of application level emulation and an aggressive strategy to determine that unpacking is complete. The advantage of application level emulation over whole system emulation is significantly greater performance. Application level emulation for automated unpacking has had commercial interest but has realized few academic publications evaluating its effectiveness and performance.

Dynamic binary instrumentation was proposed as an alternative to using an instrumented emulator employed by Renovo and pandora's bochs. Omni pack and saffron proposed automated unpacking using native execution and hardware based memory protection features. This results in high performance in comparison to emulation based unpacking. The disadvantage of unpacking using native execution is evident on e-mail gateways because a virtual machine or emulator is quire to execute the malware. A virtual machine proach to unpacking, using x86 hardware extensions, was proposed in ether. The use of such a virtual machine and equally to a whole system emulator is the requirement to install a license for each guest operating system. This restricts desktop adoption which typically has a single license. Virtual machines are also inhibited by slow start-up times, which again are problematic for desktop use. The use of a virtual machine also prevents the system being cross platform, as the guest and host cups must be the same.

B. Polymorphic Malware Classification

Malware classification has been proposed using a variety of techniques [5, 6]. A variation of n -grams, coined n-perms has been proposed [6] to describe malware char-acteristics and subsequently used in a classifier. An alter-native approach is using the basic blocks of unpacked malware, classified using edit distances, inverted indexes and bloom filters [7]. The main disadvantage of these approaches is that minor changes to the malware source code can result in significant changes to the resulting byte stream after compilation. This change can significantly impact the classification. Abstract interpretation has been used to construct tree structure signatures and model malware semantics. However, these approaches do not perform inexact matching. Windows api call sequences have been proposed [8] as an invariant characteristic, but correctly identifying the api calls can be problematic when code packing obscures the result. Flowgraph based classification is an alternative method that attempts to solve this issue by looking at control flow as a more invariant characteristic between malware variants.

Flowgraph based classification utilizes the concept of approximating the graph edit distance, which allows the construction of similarity between graphs. The commercial system vxclass presents a system for unpacking and malware classification based on similarity of flow graphs. The algorithm in vxclass is based on identifying fixed points in the graphs and greedily matching neighboring nodes [9-11]. Bin hunt provides a more thorough test of flowgraph similarity by soundly identify-ing the maximum common sub graph, but at reduced levels of performance and without application to malware classification. Identifying common sub graphs of fixed sizes can also indicate similarity and has better performance .Smith identifies malware with similar call graphs using minimum cost bipartite graph matching and the Hungarian algorithm, improving upon the greedy approach to graph matching investigated in earlier re-search. Smith additionally uses metric trees to improve performance on graph based database searches. Tree automata to quickly recognize whole program control flow graphs has also been proposed but this approach only identifies isomorphisms or sub graph isomorphism, and not approximate similarity. Context free grammars have been extracted from malware to describe control flow, but this research only identifies whole program equivalence. Clustering is related to classification and has been proposed for use on call graphs using the graph edit distance to construct similarity matrices between samples.

C. The Difference between Malwise and Previous Work

Our research differs from previous flowgraph classification research by using a novel approximate control flow graph matching algorithm employing structuring. We are the first to use the approach of structuring and decompilation to generate malware signatures. This allows us to use string based techniques to tackle otherwise infeasible Graph problems. We use an exact matching algorithm which performs in near real-time while still being able to identify approximate matches at a whole program level. The novel set similarity search we perform enables the real-time classification of malware from a large database. No prior related research has performed in real time. The classification systems in most of the previous research measure similarity in the call graph and control flow graphs, whereas our work relies entirely on the control flow graphs at a procedure level. Additionally distinguishing our work is the proposed automated unpacking system, which is integrated into the flow graph based classification system.

III. Problem Definitions and Our Approach

The problem of malware classification and variant detection is defined in this section. The problem summary is to use instance based learning and perform a similarity search over a malware database. Additionally defined in this section is an overview of our approach to design the malwise system.

A. Problem Definition

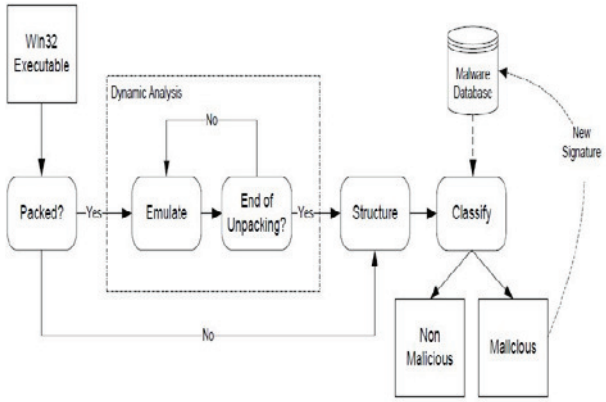
A malware classification system is assumed to have advance access to a set of known malware. This is for construction of an initial malware database. The database is constructed by identifying invariant characteristics in each malware and generating an associated signature to be stored in the database. After database initialization, normal use of the system commences. The system has as input a previously unknown binary that is to be classified as being malicious or non malicious. The input binary and the initial malware binaries may have additionally undergone a code packing transformation to hinder static analysis. The classifier calculates similarities between the input binary and each malware in the database. The similarity is measured as a real number between 0 and 1 - 0 indicating not at all similar and 1 indicating an identical or very similar match. This similarity is

a based on the similarity between malware characteristics in the data-base. If the similarity exceeds a given threshold for any malware in the database, then the input binary is deemed a variant of that malware, and therefore malicious. If identified as a variant, the database may be updated to incorporate the potentially new set of generated signatures associated with that variant.

B. Our Approach

Our approach employs both dynamic and static analysis to classify malware. Entropy analysis initially determines if the binary has undergone a code packing transformation. If packed, dynamic analysis employing application level emulation reveals the hidden code using entropy analysis to detect when unpacking is complete. Static analysis then identifies characteristics, building signatures for control flow graphs in each procedure. The similarities between the set of control flow graphs and those in a malware database accumulate to establish a measure of similarity. A similarity search is performed on the mal-ware database to find similar objects to the query.

C. Architecture Diagram



ALGORITHMS AND TECHNIQUES USED IN FEATURE WORK

Navie Bayes:

It will generate probability of no of occurrences of same malware.

J48:

It will construct binary tree and classifies true positives and true negative.

Random Forest:

It will construct multiple no of binary trees to provide analysis for large and different no of malwares.

IV. CONCLUSIONS

Malware can be classified according to similarity in its flow-graphs. This analysis is made more challenging by packed malware. In this paper we proposed different algorithms to unpack malware using application level emulation. We also proposed performing malware classification using either the edit distance between structured control flow graphs, or the estimation of isomorphism between control flow graphs. We implemented and evaluated these approaches in a fully functionally system, named Malwise. The automated un-packing was demonstrated to work against a promising number of synthetic samples using known packing tools, with high speed. To detect the completion of unpacking, we proposed and evaluated the use of entropy analysis. It was shown that our system can effectively identify variants of malware in samples of real malware. It was also shown that there is a high probability that new malware is a variant of existing malware. Finally, it was demonstrated the efficiency of unpacking and malware classification warrants Malwise as suitable for potential applications including desktop and Internet gateway and Antivirus systems.

REFERENCES

[1] F-Secure. (2007, 19 August 2009). F-Secure Reports Amount of Malware Grew by 100% during 2007.Available: http://www.fsecure.com/en_EMEA/aboutus/pressroom/news/2007/fs_news_20071204_1_eng.html
[2] K. Griffin, S. Schneider, X.Hu, and T.Chiueh, "Automatic Generation of String Signatures for Malware Detection," in Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009, Saint-Malo, France, 2009.
[3] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in 4th Virus Bulletin International Conference, 1994, pp. 178-184.
[4] J.Z.Kolter and M.A.Maloof, "Learning to detect malicious executables in the wild," in International Conference on Knowledge Discovery and Data Mining, 2004, pp. 470-478.
[5] M. E. Karim, A. Walenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," Journal in Computer Virology, vol. 1, pp. 13-23, 2005.
[6] Y. Ye, D. Wang, T. Li, and D. Ye, "IMDS: intelligent malware detection system," in Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, 2007.
[7] E. Carrera and G. Erdélyi, "Digital genome mapping–advanced binary malware analysis," in Virus Bulletin Conference, 2004, pp. 187-197.
[8] T. Dullien and R. Rolles, "Graph-based comparison of Executable Objects (English Version)," in SSTIC, 2005.
[9] I. Briones and A. Gomez, "Graphs, Entropy and Grid Computing: Automatic Comparison of Malware," in Virus Bulletin Conference, 2008, pp. 1-12.

[10] S. Cesare and Y. Xiang, "Classification of Malware Using Structured Control Flow," in 8th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010), 2010.
[11] G. Bonfante, M. Kaczmarek, and J. Y. Marion, "Morphological Detection of Malware," in International Conference on Malicious and Unwanted Software, IEEE, Alexendria VA, USA, 2008, pp. 1-8.
[12] R. T. Gerald and A. F. Lori, "Polymorphic malware detection and identification via context-free grammar homomorphism," Bell Labs Technical Journal, vol. 12, pp. 139-147, 2007.
[13] X. Hu, T. Chiueh, and K. G. Shin, "Large-Scale Malware Indexing Using Function-Call Graphs," in Computer and Communications Security,Chicago, Illinois, USA, pp. 611-620.
[14] P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee, "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," in Computer Security Applications Conference, 2006, pp. 289-300.