ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 167– 172 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

Malwise System for Packed and Polymorphic Malware

Mr. Md. Rehaman Pasha[1], Mrs. Y Prathima[2], Mr. L. Thirupati[3]

1. Assistant Professor, Dept of C.S.E. Malla Reddy Institute of Engineering Technology. rehaman17@gmail.com.
2. Assistant Professor, Dept of C.S.E. Malla Reddy Institute of Engineering Technology. prathimayenugu@yahoo.co.in
3. Assistant Professor, Dept of C.S.E. Malla Reddy Institute of Engineering Technology. thiru1274@gmail.com.

Abstract— Signature based malware detection systems have been a much used response to the pervasive problem of malware. Identification of malware variants is essential to a detection system and is made possible by identifying invariant characteristics in related samples. To classify the packed and polymorphic malware, this paper proposes a novel system, named malwise, for malware classification using a fast application level emulator to reverse the code packing transformation, and two flowgraph matching algorithms to perform classification. An exact flowgraph matching algorithm is employed that uses string based signatures, and is able to detect malware with near real-time performance. Additionally, a more effective approximate flow graph matching algorithm is proposed that uses the decompilation technique of structuring to generate string based signatures amenable to the string edit distance. We use real and synthetic malware to demonstrate the effectiveness and efficiency of Malwise. Using more than 15,000 real malware, collected from honeypots, the effectiveness is validated by showing that there is an 88% probability that new malware is detected as a variant of existing malware. The efficiency is demonstrated from a smaller sample set of malware where 86% of the samples can be classified in under 1.3 seconds.

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

1 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 previous20 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 patterns 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.

1.1 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 characteristic that is identifiable across strains of malware variants. Approximate matching 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 [14], yet approximate matching of program structure has shown to be expensive in runtime costs [15]. Poor performance in execution speed has resulted in the absence of approximate matching in end host 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 [16]. 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 [17]. Additionally, almost 50% of new malware in 2006 were repacked versions of existing malware [18].This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

2 IEEE TRANSACTIONS ON COMPUTERS

Unpacking is a necessary component to perform static analysis and to reveal the hidden characteristics of alware. In the problem scope of unpacking, it can be seen that any instances of malware utilize identical or similar packers. Many of these packers are also public, and malware often employs the u se of these public packers. Many instances of malware also employ modified versions of public packers. Being able to automatically unpack malware in any of these scenarios, in addition to unpacking novel samples, provides benefit in revealing the malware's real content – 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 [19]. 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.

2 RELATED WORKS

2.1 Automated Unpacking

Automated unpacking employing whole system emulation
was proposed in Renovo [19] and Pandora's Bochs [20]. Whole system emulation has been demonstrated to provide effective results against unknown malware samples, yet is not completely resistant to novel attacks [21].Renovo and Pandora's Bochs both detect execution of dynamically generated code to determine when unpacking is complete and the hidden code is revealed. An alternative algorithm for detecting when unpacking is complete was proposed u sing execution histograms in Hump -and-dump [22] . The Hump-and-dump was proposed as potentially desirable for integration into an emulator. Polyunpack [16] 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 [23] but has realized few academic publications evaluating its effectiveness and performance. Dynamic Binary Instrumentation was proposed as an alternative to using an instrumented emulator [24] employed by Renovo and Pandora's Bochs. Omni pack [25]and Saffron [24] 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 required to execute the malware. A virtual machine approach to unpacking, using x86 hardware extensions, was proposed in Ether [26]. 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 u se of a virtual machine also prevents the system being cross platform, as the guest and host CPUs must be the same.

2.2 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 data base.No prior related research has performed in real-time..

3 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.

3.1 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 database. 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.

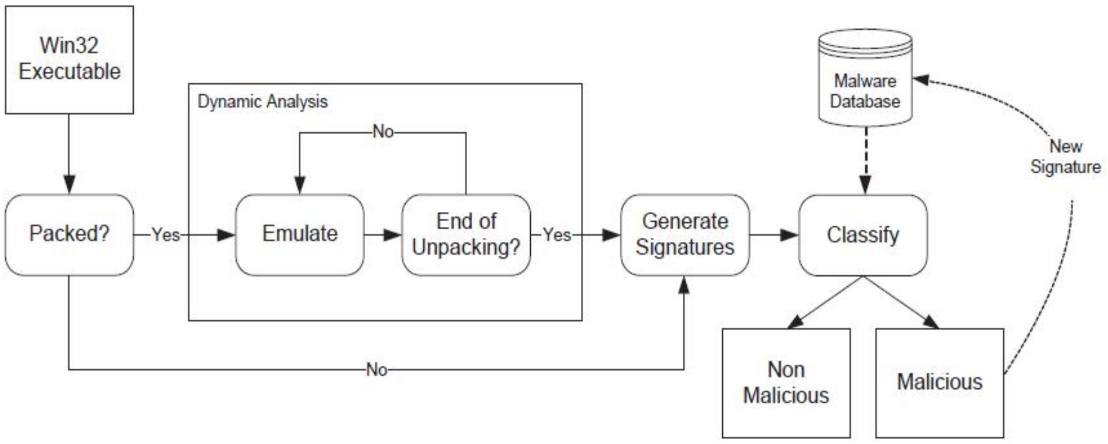


Fig. 1. Block diagram of the malware classification system

3.2 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..Two approaches are employed to generate and compare Flow graph signatures. The system design is presented in figure 1.Two flow graph matching methods are used to achieve the goal of either effectiveness or efficiency. A brief introduction is provided here.

Exact Matching: An ordering of the nodes in the control flow graph is used to generate a string based signature invariant of the flowgraph. String equality between graph invariants is used to estimate isomorphic graphs. *Approximate Matching:* The control flow graph is structured in this approach.

ISSN 2278-3091

International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages : 167– 172 (2014)
Special Issue of ICETETS 2014 - Held on 24-25 February, 2014 in Malla Reddy Institute of Engineering and Technology, Secunderabad– 14, AP, India

Structuring is the process of decompiling unstructured control flow into higher level,
Source code like constructs including structured
Conditions and iteration. Each signature representing the structured control flow is represented as a string. These signatures are then used for querying the database of known malware using an approximate dictionary search.

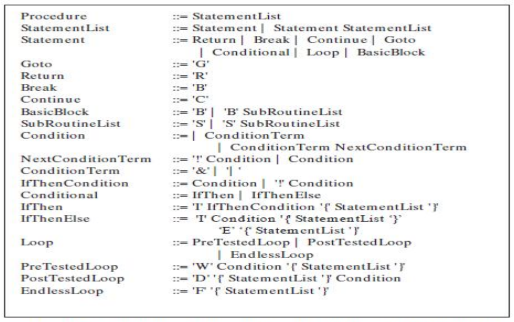
4 SYSTEM DESIGN AND IMPLEMENTATION

4.1 Identifying Packed Binaries Using Entropy Analysis

Malwise performs an initial analysis on the input binary to determine if it has undergone a code packing transformation. Entropy analysis [34], is used to identify packed binaries. The entropy of a block of data escribes the amount of information it contains. It is calculated as follows:

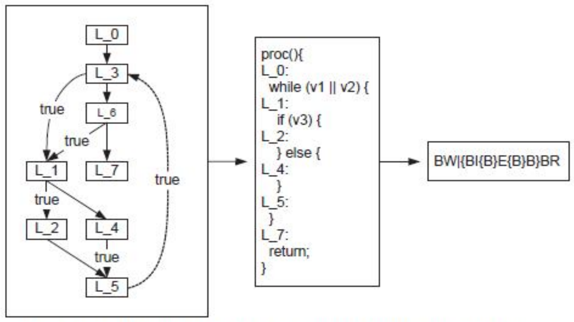
$$H(x) = -\sum_{i=1}^{N} \begin{cases} p(i)\log_2 p(i), & p(i) \neq 0 \\ 0, & p(i) = 0 \end{cases}$$

where *p(i)* is the probability of the *ith* unit of information
in event *x*'s sequence of *N* symbols. For malware packing analysis, the unit of in formation is a byte value, *N* is 256,and an event is a block of data from the malware. Compressed and encrypted data have relatively high entropy. Program code and data have much lower entropy. Packed data is typically characterized as being encrypted or compressed,



```
Procedure             ::= StatementList
StatementList         ::= Statement |  Statement StatementList
Statement             ::= Return |  Break |  Continue |  Goto
                            |  Conditional |  Loop |  BasicBlock
Goto                  ::= 'G'
Return                ::= 'R'
Break                 ::= 'B'
Continue              ::= 'C'
BasicBlock            ::= 'B' |  'B' SubRoutineList
SubRoutineList        ::= 'S' |  'S' SubRoutineList
Condition             ::= |  ConditionTerm
                              |  ConditionTerm NextConditionTerm
NextConditionTerm     ::= '!' Condition |  Condition
ConditionTerm         ::= '&' |  '| '
IfThenCondition       ::= Condition |  '!' Condition
Conditional           ::= IfThen |  IfThenElse
IfThen                ::= 'I' IfThenCondition '{' StatementList '}'
IfThenElse            ::=  'I' Condition '{ StatementList '}'
                              'E' '{' StatementList '}'
Loop                  ::= PreTestedLoop |  PostTestedLoop
                              |  EndlessLoop
PreTestedLoop         ::= 'W' Condition '{' StatementList '}'
PostTestedLoop        ::= 'D' '{' StatementList '}' Condition
EndlessLoop           ::= 'F' '{' StatementList '}'
```

Fig. 3. The grammar to represent a structured control flow graph signature



```
proc(){
L_0:
  while (v1 || v2) {
L_1:
    if (v3) {
L_2:
    } else {
L_4:
    }
L_5:
  }
L_7:
  return;
}
```

Fig. 4. The relationship between a control flow graph, a high level structured graph, and a signature.

therefore high entropy in the malware can indicate packing.

4.2 Application Level Emulation

Automated unpacking requires malware execution to be simulated so that the malware may reveal its hidden code. The hidden code once revealed is then extracted from the process image.

4.3 Entropy Analysis to Detect Completion of Hidden Code Extraction

Detection of the original entry point (OEP) during emulation identifies the point at which the hidden code is revealed and execution of the original unpacked code begins to take place. Detecting the execution of dynamic code generation by tracking memory writes was used as an estimation of the original entry point in Renovo [19].
In this approach the emulator executes the malware, and a shadow memory is maintained to track newly written memory. If any newly written memory is executed, then the hidden code in the packed binary being will now be revealed.

4.4 Static Analysis

The static analysis component of Malwise proceeds once it receives an unpacked binary. The analysis is used to extract characteristics from the input binary that can be used for classification. The characteristic for each procedure in the input binary is obtained through transforming its control flow into compact representation that is am enable to string matching. This transformation, or signature generation.

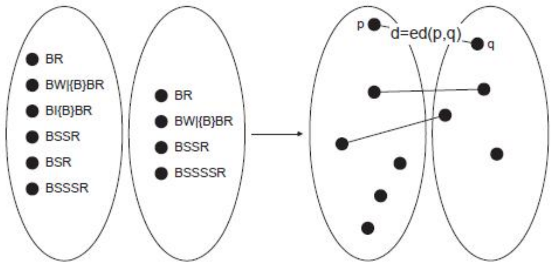


Fig. 5. Assignment of flowgraph strings between sets.

The structuring algorithm implemented in Malwise is a modified algorithm of that proposed in the DCC d ecompiler [38]. If the algorithm cannot structure the control flow graph then an unstructured branch is generated.Surprisingly, even when graphs are reducible (a measure of how inherently structured the graph is), the algorithm generates unstructured branches in a small but not insignificant number of cases. Further improvements to this algorithm to reduce the generation of unstructured branches have been proposed [39, 40]. However, these improvements were not implemented.

The grammar for a resulting signature is d efined in Fig.3.Fig. 4 shows an example of the relationship between a control flow graph, a high level structured graph, and a resulting signature.For approximate matching,a greedy assignment is made for the best approximate matching string where the similarity ratio is above 0.9. An example of assignment is shown in Fig. 5.

4.5 The Set Similarity Search

To classify the query program as malicious or benign, a similarity search is performed to find any similar malware in the database. The search can be performedexhaustively.

4.6 Complexity Analysis

We assume a search complexity is $O(\log(N))$ for both global and local flowgraph databases. The runtime complexity of malware classification is on average $O(N\log(M))$ where M is the number of control flow graphs in the database, and N is the number of control flow graphs in the input binary. N is proportional to the input binary size and not more than several hundred in most cases. The worst case can be expected to have a runtime complexity of $O(N\log(M) + AN\log(N))$, where A is the number of similar malware to the input binary. It is desirable that the malware database is not populated with a significant number of similar malware. In practice, this condition is unlikely to be significant. It is expected that the average case is processing benign samples.

4.7 Discussion

The threshold to determine if two programs are similar, in either exact flow graph matching or approximate flow graph matching, is empirically decided in Malwise. Likewise as is the similarity ratio between flow graphs. The actual figures are decided by investigating a huge number of real life malware samples. This approach is currently adopted by most Antivirus systems. It is a desirable feature that the malware classification system can adaptively select the threshold s. Machine learning based approach can be taken to achieve this. As the main focus of this research is to develop an effective and efficient system to solve the polymorphic malware problem, we leave this as
our future work.

5 EVALUATION

In this Section we describe the experiments to evaluate automated unpacking and flowgraph based classification in Malwise.

To verify our system correctly performs hidden code extraction ,we tested the prototype against 14 public packing tools. These tools perform various techniques in the resulting code packing transformation including compression, encryption, code obfuscation, debugger detection and virtual machine detection. The samples chosen to undergo the packing transformation were the Microsoft Windows XP system binaries hostname.exe and calc.exe.hostname.exe is 7680 bytes in size, and calc.exe is 114688bytes.The original entry point identified by the unpacking system was compared against what w as identified as the real OEP. To identify the real OEP, the program counter was inspected during emulation and the memory at that location examined. If the program counter was found to have the same entry point as the original binary, and the 10 bytes of memory at that location was the same as the original binary, then that address was designated the real OEP.

6 CONCLUSION

Malware can be classified according to similarity in its flowgraphs.This analysis is made more challenging by packed malware. In this paper we proposed different algorithms to unpack malware using application level emulation. We alsoproposed 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
functionaly system, named Malwise. The automated unpacking 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] Symantec, "Symantec internet security threat report: Volume XII,"Symantec2008.

[2] F-Secure. (2007, 19 August 2009). F-Secure Reports Amount ofMalware Grew by 100% during 2007. Available: http://www.fsecure.com/en_EMEA/aboutus/pressroom/news/2007/fs_news_20071204_1_eng.html

ISSN 2278-3091

[3] 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.
[4] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in *4th Virus Bulletin International Conference*, 1994, pp.178-184.
[5] 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.
[6] 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.
[7] M. Gheorghescu, "An automated virus classification system," in *Virus Bulletin Conference*, 2005, pp. 294-300.