IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 13, Issue 1 (Jul. - Aug. 2013), PP 61-66
www.iosrjournals.org

Malwise-Malware Classification and Variant Extraction

P.Nikhila[1], D.Srivalli[2], L. Ramadevi[3]
[1]*M.Tech (S.E), VCE, Hyderabad, India,*
[2]*M.Tech (S.E), VCE, Hyderabad, India,*
[3]*M.Tech (S.E), VCE, Hyderabad, India,*

Abstract :- *Malware, short for malicious software, means a variety of forms of intrusive, hostile or annoying program code or software. Malware is a pervasive problem in distributed computer and network systems. Malware variants often have distinct byte level representations while in principal belong to the same family of the malware. The byte level content is different because of small changes to the malware source code can result in significantly different compiled object code. In this project we describe malware variants with the umbrella term of polymorphism. We are the first to use the approach of structuring and decompilation to generate malware signatures. We employ both dynamic and static analysis to classify the malware. Entropy analysis was initially determines if the binary has undergone a code packing transformation. If a packed, dynamic analysis employing application level emulation reveals the hidden code using entropy analysis to detect when unpacking is complete. Static analysis is then identifies characteristics, the building signatures for control flow of graphs in each procedure. Then the similarities between the set of control flow graphs and those are in a malware database accumulate to establish a measure of similarity. A similarity search is performed on the malware database to find similar objects to the query. 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.*
Keywords*: Bayes classifier, Computer Security, Random forest, Spyware.*

I. Introduction

We are Discus about the packed and polymorphic malware, we proposes a novel system, named malwise, for malware classification using a fast application level emulator to reverse the code packing transformation, and two flow graph matching algorithms to perform classification. Now Present years, Internet worms have proliferated because of hardware and software mono-cultures, which make it possible to exploit a single vulnerability to compromise a large number of hosts. Most Internet worms follow a scan/compromise/replicate pattern of behaviour, where the worm instance first identifies possible victims, then exploits one or more vulnerabilities to compromise a host, and finally replicates there. These actions are performed through network connections and, therefore, network intrusion detection systems (NIDSs) have been proposed by the security community as mechanisms for detecting and responding to worm activity, the need for the timely generation of worm detection signatures motivated the development of systems that analyze the contents of network streams to automatically derive worm signatures. Cyber criminals constantly develop new versions of their malicious software to evade pattern-based detection by anti-virus products.

The Data security company receives it has to be determined whether the sample is Malicious or has been encountered before, it possibly in a modified form. Analogous to the human immune system, the ability to recognize commonalities among malware which belong to the same malware family would allow anti-virus products to proactively detect both known samples, as well as future releases of the malware samples from the family.

II. Related Work

We use several real vulnerabilities to create polymorphic worms; run our signature generation algorithms on workloads consisting of samples of these worms; evaluate the quality (as measured in false positives and false negatives) of the signatures produced by these algorithms; and evaluate the computational cost of these signature generation algorithms. Now Present anti-virus software typically employee a variety of methods to detect malware programs, such as signature-based scanning, heuristic-based detection, and behavioural detection

2.1 Single Polymorphic worms:

Now consider the anatomy of Polymorphic worms. We refer to a network flow containing a particular infection attempt as an instance or sample of a polymorphic worm. After briefly characterizing the types of content found in a polymorphic worm, we observe that samples of the same worm often share some invariant content due to the fact that they exploit the same vulnerability. We provide examples of real-world software vulnerabilities that support this observation. We design the SRE signature type from regular expression.

It is believed that regular expression has significant advantages for intrusion detection in terms of flexibility, accuracy, and efficiency. Regular expressions have been widely used in intrusion detection systems, for example, in Snort and Bro. However, the full regular expression is too complex and its numerous syntax rules are not needed for worm detection. Hence, we introduce the simplified regular expression as a way of representing worm signatures

2.2 Smaller signature set:

Symantec's signature set is currently tens of megabytes, and growing exponentially. If every signature were to cover many files that would significantly slow signature set growth. Hancock can do this and also generate replacements for old signatures, reducing the signature set's size. Detecting malware variants improves signature based detection methods. The size of signature databases is growing exponentially, and detecting entire families of related malicious software can prevent the blowout in the number of stored malware signatures.

Malwise is a system for detecting malware based on a using a different kind of signature. A more powerful and robust signature. In Malwise, the structure of a program is used instead of traditional string signatures that are used in Antivirus. Program structure doesn't change much when malware evolves or mutates.

Program structure can effectively fingerprint an entire family of malware and detect new family members even if they haven't been seen before by the system.

2.3 False Positive Check Index:

Illustrates the effects of accuracy and thresholds on the false positive index, with four illustrative base rates (the false positive index is the number of false positive test results for each true positive test result). It shows that increasing test accuracy makes for more attractive tradeoffs in using the test. For example, it shows that that for any base rate, if the threshold is set so as to correctly detect 50 percent of truly positive cases, or major security risks, a diagnostic with A = 0.80 has a false-positive index of about three times that of a diagnostic with A = 0.90; a diagnostic with A = 0.70 has an index of about six times that of a test with A = 0.90; and a diagnostic with A = 0.60 has an index of about eight times that of a test with A = 0.90. These ratios vary somewhat with the threshold selected, but they illustrate how much difference it would make if a high value of A could be achieved for field polygraph testing. If the diagnosis of deception could reach a level of A = 0.90, testing would produce much more attractive tradeoffs between false positives and false negatives than it has at lower levels of A.

Nevertheless, if the proportion of major security risks in the population being screened is equal to or less than 1 in 1,000, it is reasonable to expect even with optimistic assessments of polygraph test accuracy that each spy or terrorist that might be correctly identified as deceptive would be accompanied by at least hundreds of non deceptive examinees mislabeled as deceptive, from whom the spy or terrorist would be indistinguishable by polygraph test result. The possibility that deceptive examinees may use countermeasures makes this tradeoff even less attractive.

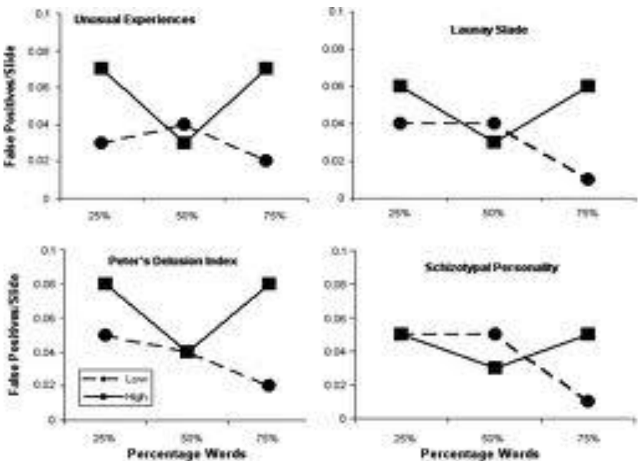


Fig1. False Positive Check Index

III. Methodology

3.1 Topology Creation:

In this module we are constructing the topology for transmitting data. Based on this topology we are testing our data transmission and data access. For creating the topology here we are using server socket to connect nodes.

3.2 Generating Signature Tree for Multiple Polymorphic Worms:

We can generate signatures of these attacks by extracting their invariant parts and any two signatures can be compared in terms of specifics according to definitions can we also organize the generated signatures into a tree so as to represent the familial resemblance of the byte sequences of these attacks. One obvious benefit of the tree organization would be that the (signature) tree would reflect and illustrate logical relations between these attacks and shed light on how worm variants evolve over time. For example, if the signature of a new attack is a child node of a known worm's signature in the signature tree, we immediately know that this new attack should be a variant of the known worm.

IV. System Implementation

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.

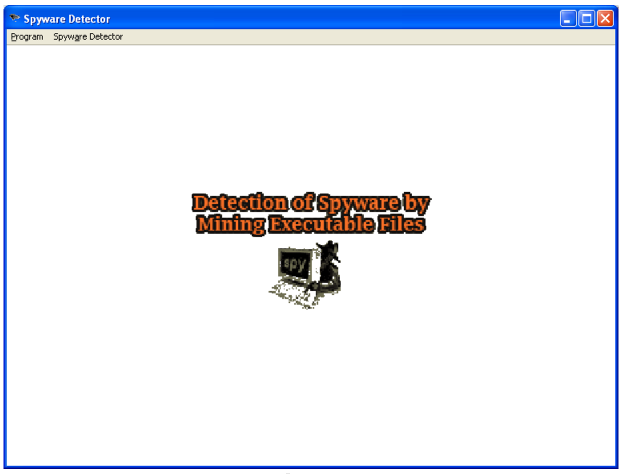
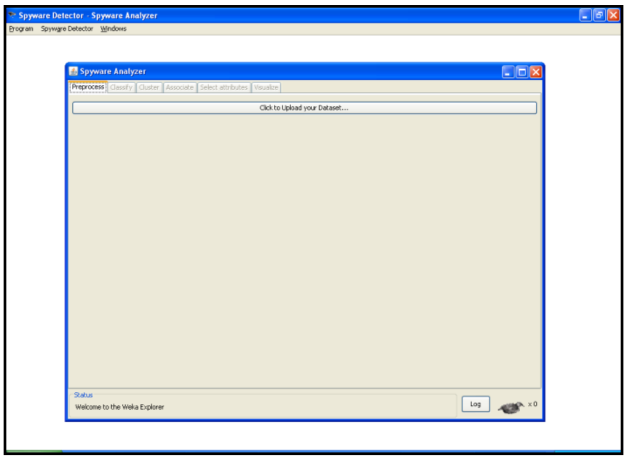


Fig 2: home Page

If not, then 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 malware database to find similar objects to the query.



Welcome to the Weka Explorer

Fig 3: upload dataset

Two approaches are employed to generate and compare flow graph signatures. Two flow graph matching methods are used to achieve the goal of either effectiveness or efficiency.

a) Exact Matching is an ordering of the nodes in the control flow graph is used to generate a string based signature or graph invariant of the flow graph.

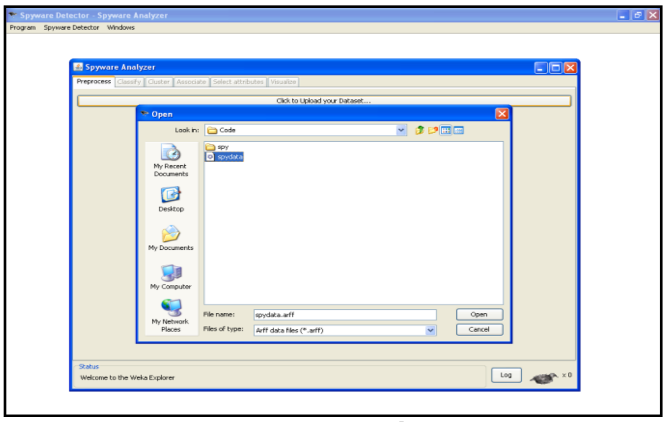
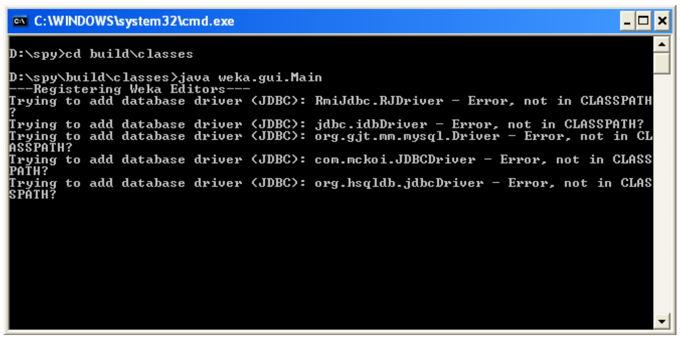


Fig 4: open spydata

b) Approximate Matching is the control flow graph is structured in this approach. Structuring is the process of decompiling unstructured control flow into higher level, source code like constructs including structured conditions and iteration.

4.1 Data Collection:

Our data set consists of 100 binaries out of which 90 are benign and 10 are spyware binaries. The benign files were collected from Download.com, which certifies the files to be free from spyware. The spyware files were downloaded from the links provided by SpywareGuide.com. This hosts information about different types of spyware and other types of malicious software.

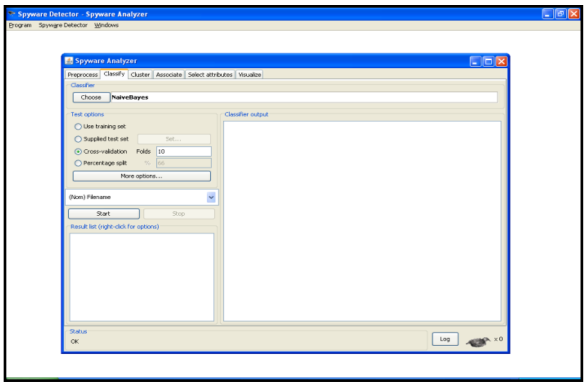

```
C:\WINDOWS\system32\cmd.exe

D:\spy>cd build\classes

D:\spy\build\classes>java weka.gui.Main
---Registering Weka Editors---
Trying to add database driver (JDBC): RmiJdbc.RJDriver - Error, not in CLASSPATH
?
Trying to add database driver (JDBC): jdbc.idbDriver - Error, not in CLASSPATH?
Trying to add database driver (JDBC): org.gjt.mm.mysql.Driver - Error, not in CL
ASSPATH?
Trying to add database driver (JDBC): com.mckoi.JDBCDriver - Error, not in CLASS
PATH?
Trying to add database driver (JDBC): org.hsqldb.jdbcDriver - Error, not in CLAS
SPATH?
```
Fig 5: run command

4.2 Byte Sequence Generation:

We have opted to use byte sequences as data set features in our experiment. These byte sequences represent fragments of machine code from an executable file. We use xxd, which is a UNIX-based utility for generating hexadecimal dumps of the binary files. From these hexadecimal dumps we may then extract byte sequences, in terms of n-grams of different sizes.

4.3 Dataset Generation:

Two ARFF databases based on frequency and common features were generated. All input attributes in the data set are represented by Booleans. These ranges are represented by either 1 or 0.

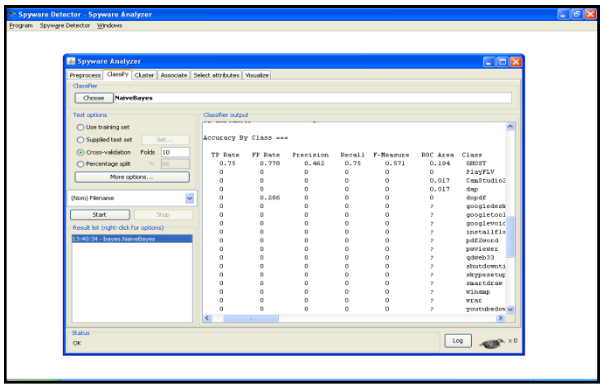

Spyware Detector - Spyware Analyzer
Program Spyware Detector Windows
Spyware Analyzer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose
Test options
Use training set
Supplied test set
Set...
Percentage split
More options...
Start
Stop
Result list (right-click for options)
Classifier output
Status
OK
Log

Classifier: NaiveBayes
Cross-validation Folds 10
(Nom) Filename

Fig 6: choose naivebayes

4.4 Feature Extraction:

The output from the parsing is further subjected to feature extraction. We extract the features by using following approaches, the Common Feature-based Extraction (CFBE) and Frequency-based Feature Extraction. The occurrence of a feature and the frequency of a feature. Both methods are used to obtain Reduced Feature Sets (RFSs) which are then used to generate the ARFF files.


Spyware Detector - Spyware Analyzer
Program Spyware Detector Windows
Spyware Analyzer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose
Test options
Use training set
Supplied test set
Percentage split
More options...
Start
Stop
Result list (right-click for options)
Classifier output
Status
OK
Log

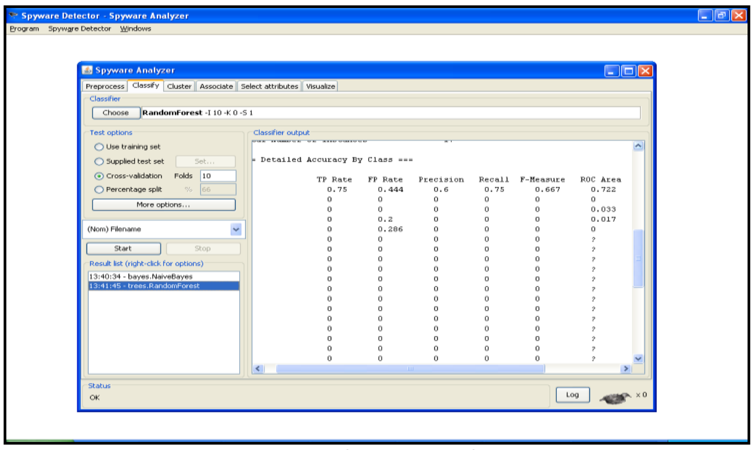
Classifier: NaiveBayes
Cross-validation Folds 10
(Nom) Filename
13:40:34 - bayes.NaiveBayes

```
Accuracy By Class ===

 TP Rate   FP Rate   Precision   Recall   F-Measure   ROC Area   Class
   0.75     0.778     0.462      0.75      0.571      0.194     GHOST
   0        0         0          0         0          0         PlayFLV
   0        0         0          0         0          0.017     CamStudio2
   0        0         0          0         0          0.017     dap
   0        0.286     0          0         0          0         dopdf
   0        0         0          0         0          ?         googledesk
   0        0         0          0         0          ?         googletool
   0        0         0          0         0          ?         googlevoic
   0        0         0          0         0          ?         installfile
   0        0         0          0         0          ?         pdf2word
   0        0         0          0         0          ?         pwviewer
   0        0         0          0         0          ?         qdweb33
   0        0         0          0         0          ?         shutdownti
   0        0         0          0         0          ?         skypesetup
   0        0         0          0         0          ?         smartdraw
   0        0         0          0         0          ?         winamp
   0        0         0          0         0          ?         wrar
   0        0         0          0         0          ?         youtubedow
```

Fig 7: test for naivebayes

4.5 Classification:

A Naive Bayes classifier is a probabilistic classifier based on Bayes theorem with independence assumptions, i.e., the different features in the data set are assumed not to be dependent of each other. This of course, is seldom true for real-life applications. Nevertheless, the algorithm has shown good performance for a wide variety of complex problems. J48 is a decision tree-based learning algorithm. During classification, it adopts a top-down approach and traverses a tree for classification of any instance. Moreover, Random Forest is an ensemble learner. In this ensemble, a collection of decision trees are generated to obtain a model that may give better predictions than a single decision tree.


Spyware Detector - Spyware Analyzer
Program Spyware Detector Windows
Spyware Analyzer
Preprocess Classify Cluster Associate Select attributes Visualize
Classifier
Choose
Test options
Use training set
Supplied test set
Set...
Percentage split
More options...
Start
Stop
Result list (right-click for options)
Classifier output
Status
OK
Log

Classifier: RandomForest -I 10 -K 0 -S 1
Cross-validation Folds 10
(Nom) Filename
13:40:34 - bayes.NaiveBayes
13:41:45 - trees.RandomForest

```
= Detailed Accuracy By Class ===

             TP Rate   FP Rate   Precision   Recall   F-Measure   ROC Area
               0.75     0.444     0.6         0.75      0.667      0.722
               0        0         0           0         0          0
               0        0         0           0         0          0.033
               0        0.2       0           0         0          0.017
               0        0.286     0           0         0          0
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
               0        0         0           0         0          ?
```

Fig 8: test for random forest

V. Conclusion

The 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 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] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in 4th Virus Bulletin International Conference, 1994, 178-184.
[2] 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, 470-478.
[3] M. E. Karim, A. Wallenstein, A. Lakhotia, and L. Parida, "Malware phylogeny generation using permutations of code," Journal in Computer Virology, vol. 1, 2005,13-23.
[4] M. Gheorghescu, "An automated virus classification system," in Virus Bulletin Conference, 2005, 294-300.
[5] 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.
[6] E. Carrera and G. Erdélyi, "Digital genome mapping–advanced binary malware analysis," in Virus Bulletin Conference, 2004, 187-197.
[7] T. Dullien and R. Rolles, "Graph-based comparison of Executable Objects (English Version)," in SSTIC, 2005.
[8] I. Briones and A. Gomez, "Graphs, Entropy and Grid Computing: Automatic Comparison of Malware," in Virus Bulletin Conference, 2008, 1-12.
[9] S. Cesare and Y. Xiang, "Classification of Malware Using Structured Control Flow," in 8th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010), 2010.
[10] 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, 1-8.
[11] R. T. Gerald and A. F. Lori, "Polymorphic malware detection and identification via context-free grammar homomorphism," Bell Labs Technical Journal, vol. 12, 2007, 139-14.
[12] X. Hu, T. Chiueh, and K. G. Shin, "Large-Scale Malware Indexing Using Function-Call Graphs," in Computer and Communications Security, Chicago, Illinois, USA, 611-620.
[13] J. Newsome and D. Song, “Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software,” Proc. 12th Ann. Network and Distributed System Security Symp, 2005.
[14] J.R. Crandall and F.T. Chong, “Minos: Control Data Attack Prevention Orthogonal to Memory Model,” Proc. 37th Ann. IEEE/ACM Int’l Symp. Micro architecture, 2004,221-232.