MALWARE ANALYSIS

Research Supervisor:
Dr. Rajshekar K

Arun R Bhat (190010007)
Newtan AG Mukhyopadhyay (190010027)

भारतीय प्रौद्योगिकी संस्थान धारवाड
Indian Institute of Technology Dharwad

Contents

1 Abstract **3**

2 Introduction **3**

3 Static Analysis of PE header files **4**

3.1 Paper 1: Detecting Unknown Malicious Executables Using PE Headers 4

3.2 Paper 2: Detection of Packed Malware 5

4 Opcode based malware analysis **6**

4.1 Paper : Improving the detection accuracy of unknown malware by partitioning the executable in groups 6

5 CFG-based Malware Analysis **7**

5.1 Implementation 8

6 Discussions and Vision **10**

1 Abstract

Malware is any computer software intended to harm the host OS or to steal sensitive data from users. Malware analysis is the study of determining the origin, functionality and the potential impact of a given malware. Rapid increase in number of malware files and their variety made it hard to manually analyse all the features present in malware. Widening field of Machine Learning has helped in order to overcome this issue and led to automation of malware analysis without much of human interventions. Hence, we made a thorough research on how to identify the files containing malware and which features assist the most in separating them from benign files. We found that static analysis with the help of Machine Learning is one of the most efficient ways for classifying files. We studied various techniques and tools for implementing the same. Finally, we started building the pipeline for static malware analysis using CFGs.

2 Introduction

Broadly, there are two ways of malware analysis: static and dynamic analysis[3]. In static analysis, the sample code is studied, various characteristics of the executable are extracted and reverse-engineered to predict the malicious activities that it might conduct. In dynamic analysis, samples are executed in different environments and their run time behaviours like changes in file system, heat generated, traffic in the network and time taken to execute are studied. It is quite expensive in terms of resources required. Also, it might damage the working of the host machine if proper care is not taken. We studied few papers that concentrated more on static analysis using PE files and got to know the number of features that help in detecting malicious nature of the code like n-grams of byte sequences, Opcode sequences, API calls and PE header files. Among these features, we mainly focused on PE header files and Opcode sequences obtained using Control Flow Graphs (CFG). Various machine learning methods used for static analysis are: Statistical methods (Naive Bayes, Bayesian Networks), Rule-based methods (C4.5, Neuro-Fuzzy), Distance-based methods (k-NN, SVM) nad Neural networks (ANN). At last, we started implementing CFG-based malware analysis and were able to extract n-grams from Opcode sequences that we found through a variant of DFS Traversal of CFGs.

3 Static Analysis of PE header files

PE (Portable Executable) file is a data structure that Microsoft has designed to wrap the information of the code for Win-32 based sytems. It is the most wildly distributed type of malicious codes.

3.1 Paper 1: Detecting Unknown Malicious Executables Using PE Headers

Surely, there should be some differences in the characteristics between malware and benign files since they are coded keeping different goals[5]. Hence, these differences can be exploited in classifying the files as malicious or benign. The model includes four steps: Attribute extraction (DUMPBIN), Attribute Binarization(IG, GR), Attribute elimmination, and Feature selection and classifier training(SVM). PE headers contain information such as target machine, number of sections in PE, data for OS to run PE, data directory, and section table. This research obtained great results for unknown virus (accuracy: 98.2%) and email worm (94%) detection but failed to perform that well in detecting trojan(84.1%) and backdoor (89.5%) when all the features were used. We learnt about the contents of the PE file and that using SVM for feature selection can help in producing very good results at a much lower cost.

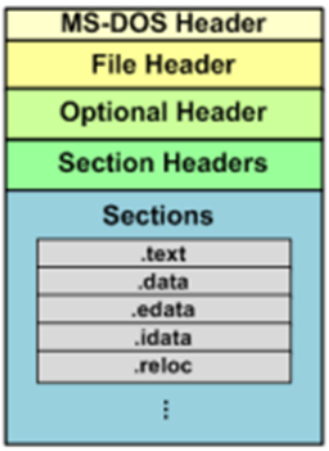


Figure 1. Portable executable file format

3.2 Paper 2: Detection of Packed Malware

Currently, Packing is the most popular obfuscation technique used by the malware writers. Packing is the process of wrapping an executable file inside another executable. It includes both compression and decompression of files[1]. The software generates completely new program on inputting an executable. On running the new executable, it unpacks the inner executable (which is the original file) into memory. About 80% malwares are packed. Multi-layer packing is also done most of the times making it much harder for detecting the malware. UPX packer is the most popular packer. It compresses all the sections (code and data) of the PE file into a single section and empties the resource section. So identifying malware in such cases should be preceded by identifying if the code is packed or not. Two most important features that help in determining if the file is packed are - the size of headers and the size of raw data. Since the unpacker code is also included in the file, header has to contain information regarding it too. So, the size of the header is generally higher. The size of raw data in packed PE file is normally NULL as all variables are uninitialized due to compression. Next step is to classify packed files as malware and benign. Size of code, size of stack reserve and Non-standard sections - are few features that help in detecting the malware. We think that features like size of code need not be that influential as benign files can also have huge codebase. Also, the paper didn't approach in direction of using NLP-based techniques on executables. We feel that PE files can contain many keywords which are different in malware files as compared to benign files, and hence can significantly improve the results.

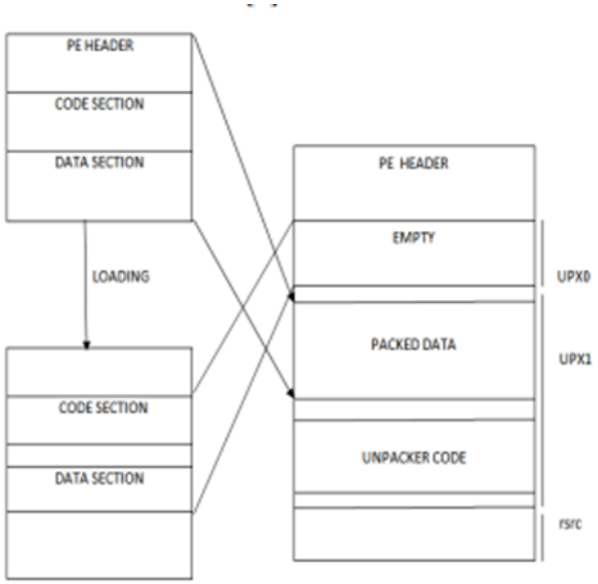


Figure 1. Functioning of UPX.

4 Opcode based malware analysis

One of the most common method of malware analysis is the opcode based analysis, where we observe and take into consideration the sequence of opcodes occuring in the executable.

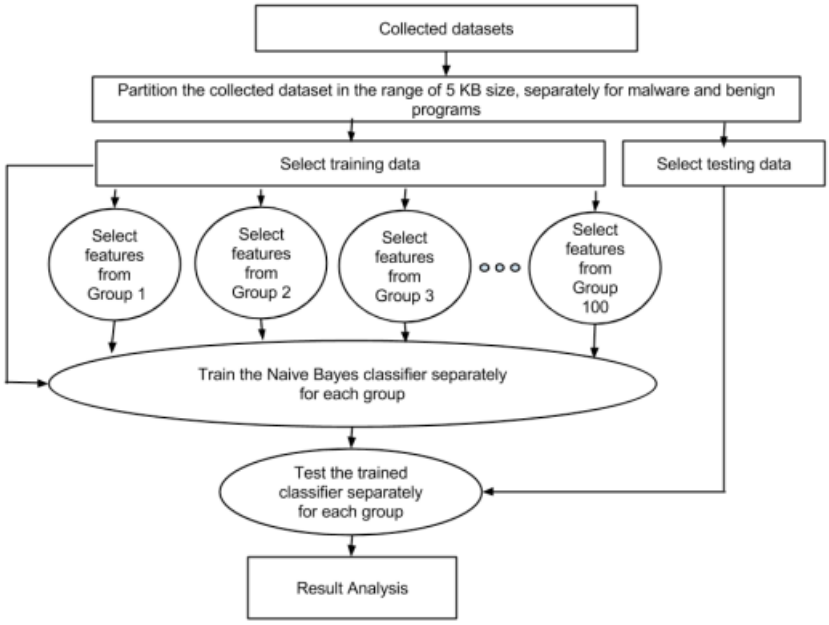
4.1 Paper : Improving the detection accuracy of unknown malware by partitioning the executable in groups

One of the most common method of Malware Analysis is observing the pattern of opcodes inside the executable by taking into consideration their occurrence patterns. The two main aspects of this paper[4] are :

- Selecting out the top features (opcodes) based on their frequency of occurence in the executable.
- Partitioning the *test* data files into smaller chunks and training and testing separately for better accuracy.

First, for both the malware as well as the benign files, we store the frequency of the opcodes separately and normalise it (by dividing the frequency of opcodes by the number of malware/benign). After that, we sort the opcodes on the basis

of the magnitude of their difference in the normalised frequency in malware and benign files and return the top n such opcodes, which serve as the feature. (Here, n is a paramater to be passed to this function which tells the number of features(opcodes) we wish to take). The dataset was divided into smaller chunks of 5KB each,and some groups of the splitted dataset were formed. After that, top features(opcodes) were extracted separately from each group by using the algorithm mentioned above. Naive Bayes Classifier was trained and tested separately for each group, which gave around 9 per cent more accuracy than regular methods.



5 CFG-based Malware Analysis

Control Flow Graphs represent how the control flows when the program is executed. These can help us in exactly determining the parts of code that are useful. The difference in sequence of execution of instructions between malware and benign files can very well be exploited in order to classify them. We started implementing the paper below and will look forward to progress much more in the next semester as we learn more.

5.1 Implementation

We can extract the opcode sequences from basic blocks of an executable. But the extra information of control flow significantly improves the learning since we can trace all the paths and mimic dynamic analysis (which is more direct way of malware analysis) without the risk of directly running the file on the host system. The proposed model has 5 steps[2]:

- Disassembly of executable (IDA)
- Generation of CFG (IDA, Radare2, BinNavi, Graph-easy)
- Feature extraction (Modified DFS, n-grams)
- Feature Scoring and Selection
- Classifier Training

By the end of this semester, we successfully completed the first three steps and are eagerly looking forward to working on next steps in the coming days. Now, we will share implementation of each step in detail along with the difficulties we faced and things we learnt as we progressed.

Disassembly of Executable

Decompiling and disassembling the exe is preliminary step before starting the analysis.

- `IDA freeware`: The free version of IDA (Interactive Disasssembler) was used by us to disassemble the executable.
- `PEId`: We have also talked about malwares being packed earlier. The PEId software is used to check for packing and the type of packer used if packed.
- `UPX unpacker`: If PEId detects positive, we use UPX unpacker to unpack the file first.

Since our project is not implemented on large datasets so far, we didn't do the unpacking step yet. But we will definitely use it once we are done with the basic steps of analysis. We successfully retrieved the list of function calls, x86 assembly code and the control flow graph in WinGraph format (.gdl file) from an executable using the IDA freeware.

Generation of Control Flow Graphs

A single binary file contains number of function calls. Extracting CFG of all of them wasn't possible in the free version of the IDA software.

- `Graph-easy`: This software was used for converting small .gdl files to .dot files which are easy to read and analyse. But converting larger files wasn't possible using this software.
- `radare2` : It is an oprn source tool that could directly give the .dot format CFGs but their readability was comparatively lower.
- `BinNavi`: This software was exclusively made for extracting CFGs, but is no longer under active development. So it uses older versions of few other dependencies and hence was not compatible with our machines. It also doesn't have much documentation on how to use.

Now we are searching for a way to either easily convert the large .gdl files to .dot files. Alternatively, we will also use python regex techniques to make .dot from radare2 more readable. We learnt about how to use these softwares, which are so diverse in nature, as per our requirements.

Feature Extraction

The generated control flow graphs information (dot files) are then fed into our python code (`extractinst.py`) to read the graph. By cleverly using the split() function, we were able to get the various details about each block -such as list of opcodes in each block, data on connection between blocks, etc. This data is then provided as input to the C++ code(`extractngram.cpp`) which builds each block and makes the graph. A modified DFS code is then run which traverses throgh all possible paths. Whenever a node is repeated in the path during DFS traversal, or whenever we reach the leaf node(i.e., a node which doesn't point to any other node), we terminate the trace and add this path to the list of paths obtained. Each path contains all the possible opcodes that we had encountered on this particular path. We use the map to replace all those numbers with the assembly instructions present in the blocks corresponding to them. Then we use the n-gram (trigram implemented, can be generalised) form of the instructions as the features for the upcoming analysis steps.

```
void dfs(node):
    if node==-1:
    return  //means that we have reached an invalid node.

    if(this path contains this node already)
    {add the block number and the instructions in sequence to the path;
    add this final path to the possible list of all paths;
    return;}

    if(node.left==-1 and node.right==-1)
    {add the block number and the instructions in sequence to the path;
    add this final path to the possible list of all paths;
    return;}

    add the block number and the instructions in sequence to the path;


    dfs(node.left);
    dfs(node.right);
```

Modified Depth First Search: Pseudo-code

6 Discussions and Vision

We started the project by reading few really good papers and got a feel for malware and how ML helps in identifying them. Idea of using control flow graphs for detecting malware interested us a lot. Hence, we dived deeper into analysis using PE header files and Opcodes as these concepts form the basis. At the end, we started implementing and went through all the difficulties as well as enjoyed the excitement of coding and exploring multiple tools. As mentioned earlier, we are done with the first three steps of implementing malware analysis using CFG - disassembly of executable, CFG generation and Feature extraction. In the coming semester, we plan to code for Feature Scoring using TF-IDF, BNS, etc. using NLP tools. Then we will conduct feature selection using statistical techniques like chisquare test, correlation, etc. Finally, we want to train the classifier using number of models like SVM, Decision Trees, etc. and observe which works the best for classifying the real world executables.

References

[1] Dhruwajita Devi and Sukumar Nandi. “Detection of packed malware”. In: Aug. 2012, pp. 22–26. DOI: 10.1145/2490428.2490431.

[2] Akshay Kapoor and Sunita Dhavale. “Control Flow Graph Based Multiclass Malware Detection Using Bi-normal Separation”. In: *Defence Science Journal* 66.2 (Mar. 2016), pp. 138–145. DOI: 10.14429/dsj.66.9701. URL: https://publications.drdo.gov.in/ojs/index.php/dsj/article/view/9701.

[3] Andrii Shalaginov et al. “Machine Learning Aided Static Malware Analysis: A Survey and Tutorial”. In: *Advances in Information Security*. Springer International Publishing, 2018, pp. 7–45. DOI: 10.1007/978-3-319-73951-9_2. URL: https://doi.org/10.1007%2F978-3-319-73951-9_2.

[4] Ashu Sharma, Sanjay Kumar Sahay, and Abhishek Kumar. “Improving the detection accuracy of unknown malware by partitioning the executables in groups”. In: *CoRR* abs/1606.06909 (2016). arXiv: 1606.06909. URL: http://arxiv.org/abs/1606.06909.

[5] Tzu-Yen Wang, Chin-Hsiung Wu, and Chu-Cheng Hsieh. “Detecting Unknown Malicious Executables Using Portable Executable Headers”. In: Jan. 2009, pp. 278–284. DOI: 10.1109/NCM.2009.385.