ABSTRACT

Malware detection developer faced a problem for a generation of new signature of malware code. A very famous and recognized technique is pattern based malware code detection technique. This leads to the evasion of signatures that are built based on the code syntax. In this paper, we discuss some well known method of malware detection based on semantic feature extraction technique. In current decade, most of authors focused on malware feature extraction process for generic detection process. The effectiveness of the Malicious Sequence Pattern Matching technique for malware detection invites for moderation and improvement of the current system and method. Some authors used rule mining technique, some other used graph technique and some also focused on feature clustering process of malware detection. To develop a malware detection software that implement machine learning to detect unknown malware. To validate that malware detection that implement machine learning will be able to achieve a high accuracy rate with low false positive rate. propose SVM classifier as detection module to identify malware. Different from the traditional k-nearest-neighbor method, SVM chooses k automatically during the algorithm process. More importantly, the SVM classifier is well-matched with the discovered sequential patterns, and is able to obtain better results than other classifiers in malware detection.

TABLE OF CONTENTS

LIST OF ABBREVATIONS

S. NO	ABBREVATION	EXPLANATION
1	**SVM**	Support Vector Machine
2	**KDD**	Knowledge Discovery in Database
3	**DWH**	Data Warehouses
4	**ODS**	Operational Data Store
5	**AMCS**	Automatic Malware Categorization System
6	**COFF**	Common Object File Format
7	**PE**	Portable Executable
8	**MSPE**	Malicious Sequential Pattern Extraction
9	**JVM**	Java Virtual Machine
10	**JDK**	Java Development Kit
11	**ICT**	In-Circuit Testing

LIST OF FIGURES

CHAPTER-1

1. INTRODUCTION:

1.1 DATA MINING:

Data mining (the analysis step of the "Knowledge Discovery in Databases" process, or KDD), a field at the intersection of computer science and statistics, is the process that attempts to discover patterns in large data sets. It utilizes methods at the intersection of artificial intelligence**,** machine learning**,** statistics, and database systems The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use Aside from the raw analysis step, it involves database and data management aspects, data preprocessing, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered structures, visualization, and online updating. Generally, data mining (sometimes called data or knowledge discovery) is the process of analyzing data from different perspectives and summarizing it into useful information - information that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified.

1.1.1 Data

Data are any facts, numbers, or text that can be processed by a computer. Today, organizations are accumulating vast and growing amounts of data in different formats and different databases. This includes:

- Operational or transactional data such as, sales, cost, inventory, payroll, and accounting
- Nonoperational data, such as industry sales, forecast data, and macro economic data

1.1.2 Information

The patterns, associations, or relationships among all this data can provide information. For example, analysis of retail point of sale transaction data can yield information on which mobile appss are selling and when.

1.1.3 Knowledge

Information can be converted into knowledge about historical patterns and future trends. For example, summary information on retail supermarket sales can be analyzed in light of promotional efforts to provide knowledge of consumer buying behavior. Thus, a manufacturer or retailer could determine which items are most susceptible to promotional efforts.

1.1.4 Data Warehouses

In computing, a data warehouse (DW or DWH) is a database used for reporting and data analysis. It is a central repository of data which is created by integrating data from multiple disparate sources. Data warehouses store current as well as historical data and are commonly used for creating trending reports for senior management reporting such as annual and quarterly comparisons. The data stored in the warehouse are uploaded from the operational systems (such as marketing, sales etc., shown in the figure to the right). The data may pass through an operational data store for additional operations before they are used in the DW for reporting. The typical ETL-based data warehouse uses staging, integration, and access layers to house its key functions. The staging layer or staging database stores raw data extracted from each of the disparate source data systems. The integration layer integrates the disparate data sets by transforming the data from the staging layer often storing this transformed data in an operational data store (ODS) database. The integrated data are then moved to yet another database, often called the data warehouse database, where the data is arranged into hierarchical groups often called dimensions and into facts and aggregate facts.

A data warehouse constructed from integrated data source systems does not require ETL, staging databases, or operational data store databases. The integrated data source systems may be considered to be a part of a distributed operational data store layer. Data federation methods or data virtualization methods may be used to access the distributed integrated source data systems to consolidate and aggregate data directly into the data warehouse database tables. Unlike the ETL-based data warehouse, the integrated source data systems and the data warehouse are all integrated since there is no transformation of dimensional or reference data. This integrated data warehouse architecture supports the drill down from the aggregate data of the data warehouse to the transactional data of the integrated source data systems.

Data warehouses can be subdivided into data marts. Data marts store subsets of data from a warehouse. This definition of the data warehouse focuses on data storage. The main source of the data is cleaned, transformed, cataloged and made available for use by managers and other business professionals for data mining, online analytical processing, market research and decision support However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform and load data into the repository, and tools to manage and retrieve metadata.

Dramatic advances in data capture, processing power, data transmission, and storage capabilities are enabling organizations to integrate their various databases into data warehouses. Data warehousing is defined as a process of centralized data management and retrieval. Data warehousing, like data mining, is a relatively new term although the concept itself has been around for years. Data warehousing represents an ideal vision of maintaining a central repository of all organizational data. Centralization of data is needed to maximize user access and analysis. Dramatic technological advances are making this vision a reality for many companies. And, equally dramatic advances in data analysis software are allowing users to access this data freely. The data analysis software is what supports data mining

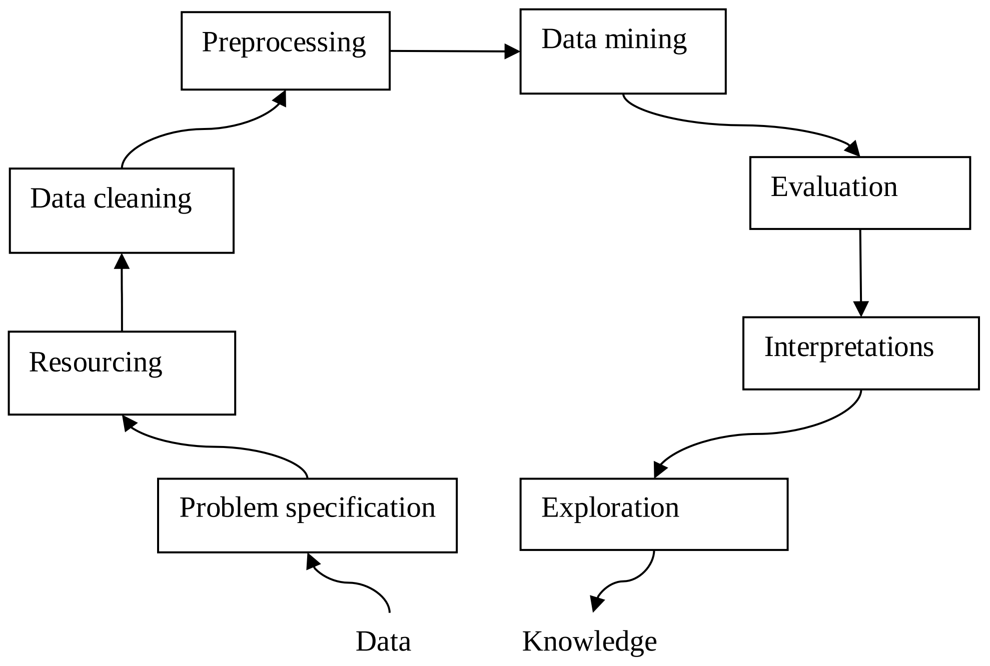


Fig 1.1 Levels of data mining

1.1.5 Data mining elements

- Extract, transform, and load transaction data onto the data warehouse system.
- Store and manage the data in a multidimensional database system.
- Provide data access to business analysts and information technology professionals.
- Analyze the data by application software.
- Present the data in a useful format, such as a graph or table.

1.1.6 Different levels of analysis

- Artificial neural networks: Non-linear predictive models that learn through training and resemble biological neural networks in structure.

- Genetic algorithms: Optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution.

- Decision trees: Tree-shaped structures that represent sets of decisions. These decisions generate rules for the classification of a dataset. Specific decision tree methods include Classification and Regression Trees (CART) and Chi Square Automatic Interaction Detection (CHAID). CART and CHAID are decision tree techniques used for classification of a dataset. They provide a set of rules that you can apply to a new (unclassified) dataset to predict which records will have a given outcome. CART segments a dataset by creating 2-way splits while CHAID segments using chi square tests to create multi-way splits. CART typically requires less data preparation than CHAID.

- Nearest neighbor method: A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset (where k 1). Sometimes called the k-nearest neighbor technique.

- Rule induction: The extraction of useful if-then rules from data based on statistical significance.

- Data visualization: The visual interpretation of complex relationships in multidimensional data. Graphics tools are used to illustrate data relationships.

1.1.7 Data Mining Techniques

There are several major data mining techniques have been developed and used in data mining projects recently including association, classification, clustering, prediction and sequential patterns. Association is one of the best known data mining technique. In association, a pattern is discovered based on a relationship of a particular item on other items in the same transaction. For example, the association technique is used in market basket analysis to identify what mobile appss that customers frequently purchase together. Based on this data businesses can have corresponding marketing campaign to sell more mobile appss to make more profit.

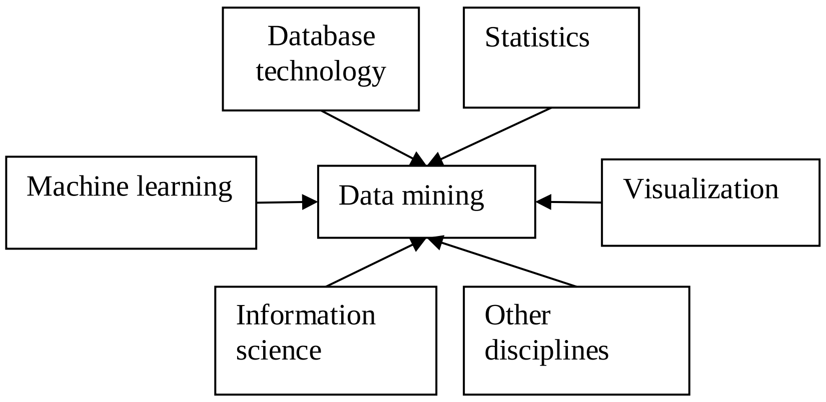


Fig 1.2 Techniques of data mining

1.1.8 Classification

Classification is a classic data mining technique based on machine learning. Basically classification is used to classify each item in a set of data into one of predefined set of classes or groups. Classification method makes use of mathematical techniques such as decision trees, linear programming, neural network and statistics. In classification, make the software that can learn how to classify the data items into groups. For example, can apply classification in application that “given all past records of employees who left the company, predict which current employees are probably to leave in the future.” In this case, divide the employee’s records into two groups that are “leave” and “stay”.

1.1.9 Clustering

Clustering is a data mining technique that makes meaningful or useful cluster of objects that have similar characteristic using automatic technique. Different from classification, clustering technique also defines the classes and put objects in them, while in classification objects are assigned into predefined classes. To make the concept clearer, can take library as an example.

1.2 MACHINE LEARNING

Machine learning (ML)is the study of computer algorithms thatimprove automatically through experience. Itis seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop conventional algorithms to perform the needed tasks.

In its application across business problems, machine learning is also referred to as predictive analytics Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system.

1.2.1 Problem Statement

With the growth of technology, the number of malware are also increasing day by day. Malware now are designed with mutation characteristic which causes an enormous growth in number of the variation of malware. Not only that, with the help of automated malware generated tools, novice malware author is now able to easily generate a new variation of malware. With these growths in new malware, traditional signature based malware detection are proven to be ineffective against the vast variation of malware. On the other hand, machine learning methods for malware detection are proved effective against new malwares. At the same time, machine learning methods for malware detection have a high false positive rate for detecting malware.

1.2.2 Objective

To investigate on how to implement machine learning to malware detection in order to detection unknown malware. To develop a malware detection software that implement machine learning to detect unknown malware.

CHAPTER-2
LITERATURE SURVEY

2.1 Title: DLL Miner: structural mining for malware detection.

Author: Masoud Narouei, Mansour Ahmadi.

Existing anti-malware products usually use signature-based techniques as their main detection engine. Although these methods are very fast, they are unable to provide effective protection against newly discovered malware or mutated variant of old malware. Heuristic approaches are the next generation of detection techniques to mitigate the problem. These approaches aim to improve the detection rate by extracting more behavioral characteristics of malware. Although these approaches cover the disadvantages of signature-based techniques, they usually have a high false positive, and evasion is still possible from these approaches. In this paper, we propose an effective and efficient heuristic technique based on static analysis that not only detect malware with a very high accuracy, but also is robust against common evasion techniques such as junk injection and packing. Our proposed system is able to extract behavioral features from a unique structure in portable executable, which is called dynamic-link library dependency tree, without actually executing the application.

2.2 Title: An intelligent PE-malware detection system based on association mining.

Author: Yanfang Ye · Dingding Wang · Tao Li.

The proliferation of malware has presented a staid threat to the security of computer systems. Traditional signature-based anti-virus systems fail to identify polymorphic metamorphic and new, previously unseen malicious executables. Data mining methods such as Naive Bayes and Decision Tree contains studied on small collections of executables. In this paper, hidden on the analysis of Windows APIs called by PE files, we extend the Intelligent Malware Detection System (IMDS) using Objective-Oriented Association mining based classification. IMDS is an integrated structure consisting of three major modules: PE parser, OOA rule maker, and rule based classifier. An OOA_Fast_FP-Growth algorithm is adapted to competently generate OOA rules for classification. A broad experimental study on a large group of PE files obtained from the anti-virus laboratory of King Soft Corporation is performed to evaluate various malware detection approaches. Promising investigational results demonstrate that the precision and efficiency of our IMDS system outperform popular anti-virus software such as Norton Antivirus and McAfee Virus Scan, as well as preceding data mining based finding systems which employed Naive Bayes, Support Vector Machine (SVM) and Decision Tree techniques.

2.3 Title: Data Mining Methods for Detection of New Malicious Executables.

Author: Matthew G. Schultz and Eliza Eskin

A serious safety threat today is malicious executables, especially new, unseen malicious executables often arriving as email attachments. These new-fangled malicious executables are formed at the rate of thousands every year and pose a serious safety threat. Current anti-virus systems attempt to detect these new malicious programs with heuristics generated by hand. This approach is costly and oftentimes hopeless. In this paper, we present a data-mining framework that detects new, before unseen malicious executables accurately and mechanically. The data-mining framework mechanically found patterns in our data set and second-hand these patterns to detect a set of new malicious binaries. Comparing our finding methods with a usual signature based method; our method more than doubles the current detection rates for new malicious executables. Data, such as byte code, and use these patterns to detect future instances in similar data. Our framework uses classifiers to identify new malicious executables. A classifier is a rule set, or detection model, generated by the data mining algorithm that was skilled over a given set of training data.

2.4 Title: Automatic Malware Categorization Using Cluster Ensemble.

Author: Yanfang Ye, Tao Li

Malware categorization is significant problem in malware analysis and has attracted a lot of attention of computer safety researchers and anti-malware industry recently. Today’s malware samples are created at a rate of millions per day with the expansion of malware writing techniques. There is thus an urgent need of effective methods for routine malware categorization. The last few years, many clustering techniques have been employed for automatic malware classification. However, such techniques have isolated successes with partial effectiveness and efficiency, and few have been applied in actual anti-malware industry. In this paper, resting on the analysis of instruction frequency and function-based instruction sequences, we increase an Automatic Malware Categorization System (AMCS) for mechanically grouping malware samples into families that share some common distinctiveness using a cluster ensemble by aggregating the clustering solutions generated by dissimilar base clustering algorithms.

2.5 Title: Malware Analysis and Detection Using Machine Learning Algorithms.

Author : Tao Feng

In this paper, we describe our research effort on malware detection based on window API calls. We develop an integrated IMDS system consisting of PE parser, OOA rule generator and rule based classifier. First, a PE parser is developed to extract the Windows API execution calls for each Windows portable executable. Then, the OOA_Fast_FP-Growth algorithm is adapted to generate association rules with the objectives of finding malicious and benign executables. This algorithm achieves much higher efficiency than previous OOA mining algorithms. Finally, classification is performed based on the generated rules. Experiments on a large real data collection from anti-virus lab at Kingsoft Corp. demonstrate the strong abilities of our IMDS system to detect polymorphic and new viruses. And the efficiency, accuracy and the scalability of our system outperform other current widely used anti-virus software and other data mining based malware detection methods. Our work results in a real malware detection system, which has been incorporated into the scanning tool of KingSoft’s AntiVirus software.

2.6 Title: Automatic Malware Categorization Using Cluster Ensemble.

Author: Yanfang Ye, Tao Li

Scanning files for signatures is a established technology, but exponential growth in unique malware programs has caused an detonation in signature database sizes. One solution to this problem is to use string signatures, each of which is a next byte sequence that potentially can match many variants of a malware family. However, it is not clear how to routinely generate these string signatures with a sufficiently low false positive rate. Hancock is the first string signature creation system that takes on this challenge on a large scale. To minimize the false positive rate, Hancock features a scalable model that estimates the occurrence possibility of arbitrary byte sequences in good ware programs, a set of library code recognition techniques, and diversity-based heuristics that ensure the contexts in which a signature is implanted in containing malware files are similar to one another. With these techniques shared, Hancock is able to mechanically generate string signatures with a false positive rate below 0.1%.

2.7 A feature extraction and selection tool for android malware detection.

Author :Y. Chen

In this paper, resting on the analysis of instruction frequency and function-based instruction sequences of the Windows Portable Executable (PE) files, we develop AMCS for automatically grouping malware samples into families that share some common characteristics using a cluster ensemble by aggregating the clustering solutions generated by different base clustering algorithms. To overcome the instability of clustering results and improve clustering performance, our AMCS system use a cluster ensemble to aggregate the clustering solutions generated by different algorithms. However, such techniques have isolated successes with partial effectiveness and efficiency, and few have been applied in actual anti-malware industry. In this paper, resting on the analysis of instruction frequency and function-based instruction sequences, we increase an Automatic Malware Categorization System (AMCS) for mechanically grouping malware samples into families that share some common distinctiveness using a cluster ensemble by aggregating the clustering solutions generated by dissimilar base clustering algorithms. We develop new base clustering algorithms to account for the different characteristics of feature representations and propose a novel cluster ensemble framework for combining individual clustering solutions. We show that the domain knowledge in the form of sample-level constraints can be naturally incorporated in the ensemble framework. To the best of our knowledge, this is the first work of applying such cluster ensemble methods for malware categorization.

2.8 Malware detection by behavioural sequential patterns.

Author :M. Ahmadi,

To address the weaknesses of the signature-based method, there have been many heuristic-based efforts in both the static and dynamic analysis sectors. These works have used static analysis of Un changeable executable characteristics or distance-based signature matching. Alternatively, there is dynamic analysis using techniques such as graph-based signatures and instruction sequence mining. Because of the uncertainty in these approaches, the challenge is to perfect the models by extracting more semantics or gaining a better detection rate associated with a low false alarm production. And the efficiency, accuracy and the scalability of our system outperform other current widely used anti-virus software and other data mining based malware detection methods.

2.9 Opcodes histogram for classifying metamorphic portable executable small ware.

Author :B. Rad,

The main purpose of the proposed methodology is to find a histogram of opcodes for each family, as a feature. This histogram presents the average distribution of instructions opcodes in the virus family. To achieve this purpose, firstly, we build a database of different variants of the morphed virus. Then, we extract the opcodes from these files. It can be performed using a dis-assembler program, which is able to analyze PE binaries and recognize the first byte of each instruction. In Portable

Executable, or Common Object File Format (COFF), headers are created of a COFF header, an Optional header, an MS-DOS stub, and a file signature. Using this information, we can take out the code segment of the file. By analyzing the code segment, we can separate the machine instructions. The first byte of instructions is important for us. We ignore next bytes of instructions, because they are optional bytes, generally one or more operands, upon on the operation of instructions. These bytes are usually mutated by the metamorphic engines, so they are different in various instances of a morphed family malware.

2.10 Using convolutional neural networks for classification of malware represented as images.

Author : Gibert, D

A portable executable (PE) depends on some DLLs for execution, and each DLL has a relationship with other DLLs for completing the task. This hierarchical dependency between PE and DLLs is known as DLL dependency tree. In this paper, we will propose a hybrid system that by statically extracting the DLL dependency tree from the PE, has the advantages of both static and dynamic techniques. In Portable Executable, or Common Object File Format (COFF), headers are created of a COFF header, an Optional header, an MS-DOS stub, and a file signature. However, such techniques have isolated successes with partial effectiveness and efficiency, and few have been applied in actual anti-malware industry. In this paper, resting on the analysis of instruction frequency and function-based instruction sequences, we increase an Automatic Malware Categorization System (AMCS) for mechanically grouping malware samples into families that share some common distinctiveness using a cluster ensemble by aggregating the clustering solutions generated by dissimilar base clustering algorithms. Using this information, we can take out the code segment of the file. By analyzing the code segment, we can separate the machine instructions.

CHAPTER-3
SYSTEM STUDY

3.1 Existing System

Due to its damage to Internet security, malware (e.g., virus, worm, Trojan) and its finding has caught the attention of both anti malware industry and researchers for decades. To protect genuine users from the attacks, the most significant line of defense against malware is anti-malware software products, which mostly use signature-based method for detection. However, this method fails to recognize new, unseen malicious executable. To solve this problem, in this paper, based on the instruction sequences extracted from the file sample set, we propose an effective sequence mining algorithm to discover malicious sequential patterns, and then J48 classifiers constructed for malware detection based on the discovered patterns.

The developed data mining framework composed of the proposed sequential pattern mining method and J48 classifier can well characterize the malicious patterns from the collected file samples to effectively detect newly unseen malware samples. More importantly, the SVM classifier is well-matched with the discovered sequential patterns, and is able to obtain better results than other classifiers in malware detection. To conduct a series of experiments to evaluate each part of our framework and the whole system based on real sample collection, containing both malicious and benign PE files. This strategy greatly enhances the efficiency of our algorithm.

SVM classifier for malware detection : We propose SVM classifier as detection module to identify malware. Different from the traditional k-nearest-neighbor method, SVM chooses k automatically during the algorithm process. More importantly, the SVM classifier is well-matched with the discovered sequential patterns, and is able to obtain better results than other classifiers in malware detection. To conduct a series of experiments to evaluate each part of our framework and the whole system based on real sample collection, containing both malicious and benign PE files.

3.1.1 Disadvantages

- There is no mechanism to predict the unseen malware file.
- Traditional signature-based anti-virus systems fail to detect unseen malicious executables.
- The J48 method learns over sequences of a fixed length.
- It does not find the type of malware file.

3.2 Proposed System

Sequence mining algorithm to discover malicious sequential patterns based on the machine instruction sequences extracted from the Windows Portable Executable (PE) files, then use it to construct a data mining frame work, called MSPMD (**M**alicious **S**equential **P**attern based **M**alware **D**etection), to detect new malware samples. The main contributions of this paper can be summarized as follows: Instruction sequences are extracted from the PE (Portable Executable) files as the preliminary features, based on which the malicious sequential patterns are mined in the next step.

The extracted instruction sequences can well indicate the potential malicious patterns at the microlevel. In addition, such kind of features can be easily extracted and used to generate signatures for the traditional malware detection systems. An effective sequential pattern mining algorithm, called MSPE (**M**alicious **S**equential **P**attern **E**xtraction), to discover malicious sequential patterns from instruction sequence. MSPE introduces the concept of objective- oriented to learn patterns with strong abilities to distinguish malware from benign files. Moreover, we design a filtering criterion in MSPE to filter the redundant patterns in the mining process in order to reduce the costs of processing time and search space.

This strategy greatly enhances the efficiency of our algorithm. SVM classifier for malware detection: We propose SVM classifier as detection module to identify malware. Different from the traditional k-nearest-neighbor method, SVM chooses k automatically during the algorithm process. More importantly, the SVM classifier is well-matched with the discovered sequential patterns, and is able to obtain better results than other classifiers in malware detection. To conduct a series of experiments to evaluate each part of our framework and the whole system based on real sample collection, containing both malicious and benign PE files. The results show that MSPMD is an effective and efficient solution in detecting new malware samples. The main contributionsof this paper can be summarized as follows: Instruction sequences are extracted from the PE (Portable Executable) files as the preliminary features, based on which the malicious sequential patterns are mined in thenextstep.

3.2.1 Advantages

- A Classification is based on the generated rules.
- It is able to automatically generate string signatures.
- It signatures are very useful for malware detection.
- To simulate the task of detecting new malicious executables.

CHAPTER-4

SYSTEM REQUIREMENTS

Hardware Requirements:

• System	: Pentium IV 2.4 GHz.
• Hard Disk	: 40 GB.
• Monitor	: 15 VGA Color.
• Mouse	: Logitech.
• Ram	: 512 Mb.

Software Requirements:

• Operating system	: Windows XP.
• Coding Language	: Java

4.1 Software Description:

4.1.1 Java

Java is an object oriented programming language. Java is a small, simple, safe, object oriented, interpreted or dynamically optimized, byte coded, architectural, garbage collected, multithreaded programming language with a strongly typed exception-handling for writing distributed and dynamically extensible programs.

- It is simple and object oriented.
- It allows the programmers to create user friendly interfaces.
- It is very dynamic.

- Multithreading.
- Platform independent language.
- Provides security and robustness.
- Provides support for internet programming

4.2 Primary Goals

The five primary goals of the creation of the Java language are:

- The object-oriented programming methodology should be used.
- The same program should be allowed to execute on multiple operating systems.
- The built-in support should be provided for using computer networks.
- The code from remote sources should be executed securely.
- It should be easy to use by combining the good parts of other object-oriented languages.

4.3 Different "Editions" Of The Platform

- Java ME (Micro Edition): Defines different sets of libraries (known as profiles) for devices which are sufficiently limited that supplying the full set of Java libraries would take up unacceptably large amounts of storage.
- Java SE (Standard Edition): to offer general purpose use on desktop PCs, servers and other similar devices.
- Java EE (Enterprise Edition): Java SE and various APIs are useful for multi-tier client-server enterprise applications.

The important components in the platform are the libraries, the Java compiler, and the runtime environment where Java intermediate byte code is executed.

4.4 Java Virtual Machine

The virtual machine concept that executes Java byte code programs is the important part of Java platform. The byte code generated by the compiler is the same for every system regardless of the operating system or hardware in the system that executes the program. The JIT compiler is in the Java Virtual Machine (JVM). At run-time the Java byte code is translated into native processor instructions. The translation is done by JIT compiler. It caches the native code in memory during execution.

4.5 JVM Linker

- The JVM linker is used to add the compiled class or interface to the runtime system.
- It creates static fields and initializes them.
- And it resolves names. That is it checks the symbolic names and replaces it with the direct references.

4.6 JVM Verifier

- The JVM verifier checks the byte code of the class or interface before it is loaded.
- If any error occurs then it throws Verify Error exception.

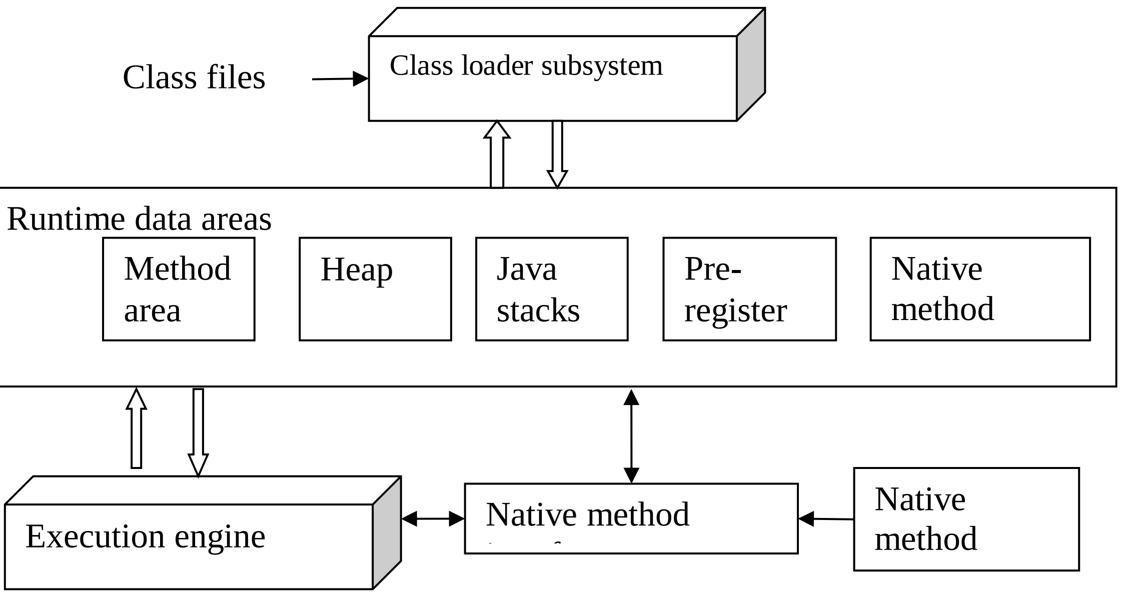


Fig 4.1 JVM Architecture

4.7. Class Libraries

Most of the modern operating systems provide a large set of reusable code to simplify the job of the programmer. This code is actually provided as a set of dynamically loadable libraries that can be called at runtime by the applications. Java Platform is not dependent on any specific operating system so the applications are not rely on any of the existing libraries. The Java Platform provides a set of standard class libraries which contains most of the same reusable functions commonly found in modern operating systems.

The Java class libraries provide three purposes within the Java Platform. They provide the programmer a well-known set of functions to perform common tasks like other standard code libraries. The class libraries provide an abstract interface to tasks that would normally depend heavily on the hardware and operating system. Tasks such as file access and network access are heavily dependent on the native capabilities of the platform. The required native code is implemented internally by the Java java.io and java.net libraries, and then it provides a standard interface for the Java applications to perform the file access and network access. If the underlying platform does not support all of the features a Java application expects, then the class libraries can either emulate those features or at least provide a consistent way to check for the presence of a specific feature.

4.8 Platform Independence

Platform independence allows the programs written in the Java language to run same on any provided hardware or operating-system platform. Using Java language programmer writes a program once, compile the code once, and it can be run anywhere.

This can be achieved by the Java compilers. The Java compilers compile the Java language code halfway. Then the code is executed on a virtual machine (VM). Virtual machine is a program written in native code on the host hardware. It interprets and executes the generic Java byte code. The features of the host machine can be accessed using the standardized libraries. The JIT compiler interprets the byte code into native machine code. The first implementations of the language used an interpreted virtual machine to achieve portability. These implementations produced programs that ran more slowly than programs compiled to native executables, for instance written in C or C++, so the language suffered a reputation for poor performance. More recent JVM implementations produce programs that run significantly faster than before, using multiple techniques.

The technique, called as just-in-time compilation (JIT), translates the Java byte code into native code at run-time. This results in a fast execution of a program that ex than interpreted code. This technique results in compilation overhead during the execution. Dynamic recompilation is used by most of the modern virtual machines. The critical parts of the program are analyzed by the virtual machines to capture the behavior of the program running. Then the particular parts are recompiled and optimized. The optimizations achieved by dynamic recompilation are more efficient than static compilation. The reason is the dynamic compiler optimizes the code based on the runtime environment characteristics and the set of classes. Also it can identify .The critical parts of the program. Both the JIT compilation and dynamic recompilation makes the Java programs to achieve the speed of native code without losing portability.

The other technique, called static compilation, is used to compile the native code like other traditional compilers. Static Java compiler, like GCJ, translates the Java language code into native object code. It removes the intermediate byte code stage. This results good performance compared to interpretation, but it needs portability. The output of the static compilers can only be run on a single architecture.

4.9 Java Runtime Environment

The applications deployed on the Java Platform can be executed using the software Java Runtime Environment, sometimes called as JRE. Usually the end-users use a JRE in software packages and Web browser plugins. Also the superset of the JRE called the Java 2 SDK (more commonly known as the JDK) is provided by Sun. Java 2 SDK includes development tools like the Java compiler, Javadoc, Jar and debugger.

The runtime engine is provides the automated exception handling tools to handle the exceptions occur in the system. The runtime engine captures the debugging information when an exception was thrown.
The critical parts of the program are analyzed by the virtual machines to capture the behavior of the program running. Then the particular parts are recompiled and optimized. The optimizations achieved by dynamic recompilation are more efficient than static compilation. The reason is the dynamic compiler optimizes the code based on the runtime environment characteristics and the set of classes. Also it can identify the critical parts of the program.

CHAPTER-5
MODULES DESCRIPTION

MODULES

- Data samples Acquisition
- Instruction Sequence Extractor
- Malicious Sequential Pattern Miner
- SVM Classifier

5.1 Data Sample Acquisition

In this module is used to input to the system. It contains the malicious and benign files. In the training phase, a classifier is generated malicious sequence pattern mining based malicious and benign.

5.2 Instruction Sequence Extractor

Instruction sequences are extracted from the PE (Portable Executable) files as the preliminary features, based on which the malicious sequential patterns are mined in thenextstep. The extracted instruction sequences can well indicate the potential malicious patterns at the microlevel. In addition, such kind of features can be easily extracted and used to generate signatures for the traditional malware detection systems.

5.3 Malicious Sequential Pattern Miner

An effective sequential pattern mining algorithm, called MSPE (Malicious **S**equential **P**attern **E**xtraction), to discover malicious sequential patterns from instruction sequence. MSPE introduces the concept of objective- oriented to learn patterns with strong abilities to distinguish malware from benign files. Moreover, we design a filtering criterion in MSPE to filter the redundant patterns in the mining process in order to reduce the costs of processing time and search space.

5.4 SVM Classifier

It discovers malicious sequential patterns based on the machine instruction sequences extracted from the Windows Portable Executable (PE) files, then use it to assemble a data mining framework, called MSPMD (Malicious Sequential Pattern based Malware Detection), to detect new malware samples, And then Support Vector Machine (SVM) classifier is constructed for malware detection based on the discovered patterns. The other technique, called static compilation, is used to compile the native code like other traditional compilers. Static Java compiler, like GCJ, translates the Java language code into native object code. It removes the intermediate byte code stage. This results good performance compared to interpretation, but it needs portability. The output of the static compilers can only be run on a single architecture. The technique, called as just-in-time compilation (JIT), translates the Java byte code into native code at run-time. This results in a fast execution of a program that ex than interpreted code. This technique results in compilation overhead during the execution. Dynamic recompilation is used by most of the modern virtual machines. The critical parts of the program are analyzed by the virtual machines to capture the behavior of the program running. Then the particular parts are recompiled and optimized. The optimizations achieved by dynamic recompilation are more efficient than static compilation.

CHAPTER-6

SYSTEM DESIGN

6.1 ARCHITECTURE DIAGRAM

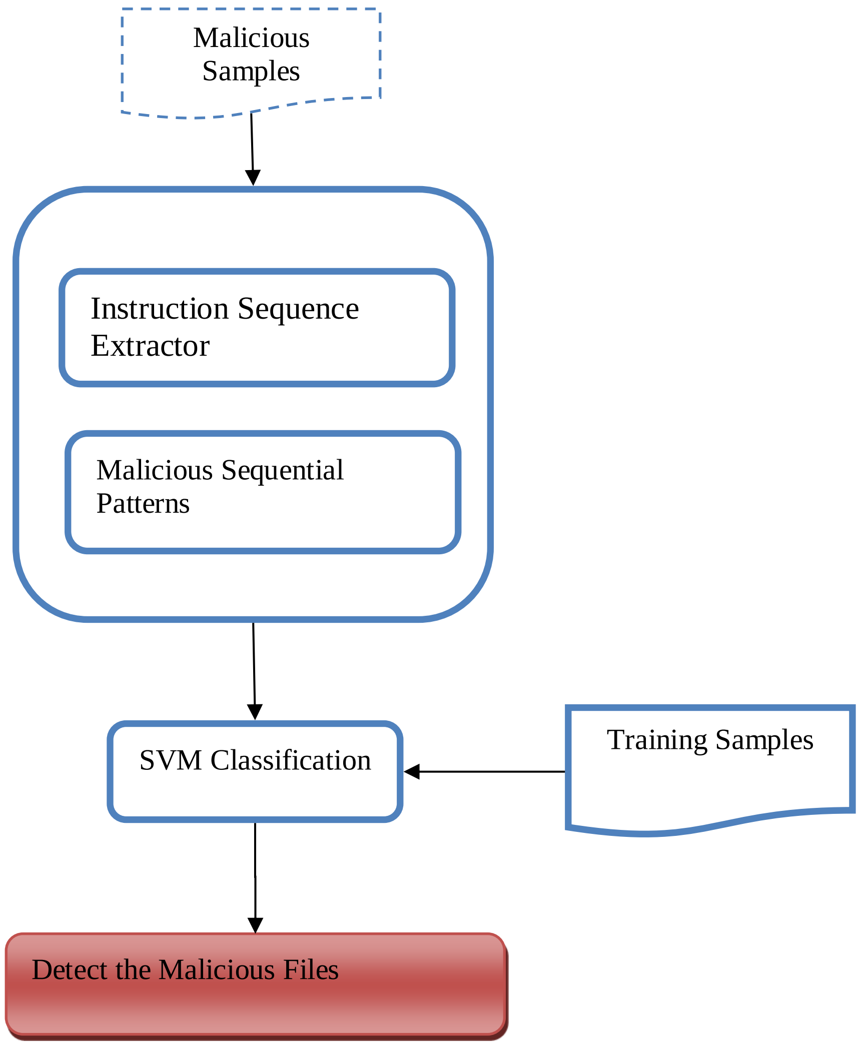

Malicious Samples
Instruction Sequence Extractor
Malicious Sequential Patterns
SVM Classification
Training Samples
Detect the Malicious Files

Fig 6.1 Architecture Diagram

CHAPTER-7
SYSTEM TESTING

7.1 Static vs dynamic testing

There are many approaches available in software testing. Reviews, walkthroughs, or inspections are referred to as static testing, whereas actually executing programmed code with a given set of test cases is referred to as dynamic testing. Static testing is often implicit, as proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run. Dynamic testing may begin before the program is 100% complete in order to test particular sections of code and are applied to discrete functions or modules.

Static testing involves verification, whereas dynamic testing involves validation. Together they help improve software quality. Among the techniques for static analysis, mutation testing can be used to ensure the test-cases will detect errors which are introduced by mutating the source code.

7.2 The box approach

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

7.2.1 White-Box testing

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases.

The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

100% statement coverage ensures that all code paths or branches (in terms of control flow) are executed at least once. This is helpful in ensuring correct functionality, but not sufficient since the same code may process different inputs correctly or incorrectly.

7.2.2 Black-box testing

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

7.2.3 Grey-box testing

Grey-box testing (American spelling: gray-box testing) involves having knowledge of internal data structures and algorithms for purposes of designing tests, while executing those tests at the user, or black-box level. The tester is not required to have full access to the software's source code. Manipulating input data and formatting output do not qualify as grey-box, because the input and output are clearly outside of the "black box" that we are calling the system under test.

This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed.

7.3 Testing Levels

There are generally four recognized levels of tests: unit testing, integration testing, system testing, and acceptance testing. Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

7.3.1 Unit testing

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Rather than replace traditional QA focuses, it augments it.

Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.

7.3.2 Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

7.3.3 System testing

System testing, or end-to-end testing, tests a completely integrated system to verify that it meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

In addition, the software testing should ensure that the program, as well as working as expected, does not also destroy or partially corrupt its operating environment or cause other processes within that environment to become inoperative.

7.3.4 Installation testing

An installation test assures that the system is installed correctly and working at actual customer's hardware.

7.3.5 Software performance testing

Performance testing is generally executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload.

It can also serve to investigate measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

7.3.6 Usability testing

Usability testing is to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application.

7.3.7 Validation Testing

Lot of module are available in the billing system, there are define in below concepts

In this software each module and particularly every, entity it should be tested, the particular input it could be got a correct input and correct out put it should be showed, at the same time the mobile no's only got ten digits only more than values not be supported.

The report module and upload module and the user register module every module and this activity, performance to be tested in every entity successfully tested.

CHAPTER-8

CONCLUSION & FUTURE ENHANCEMENT

8.1 Conclusion

To develop a data-mining- based detection framework called Malicious Sequential Pattern based Malware Detection (MSPMD), which is composed of the proposed sequential pattern mining algorithm (MSPE) and SVM classifier. It first extracts instruction sequences from the PE file samples and conducts feature selection before mining; then MSPE is applied to generate malicious sequential patterns. For the testing file samples, after feature representation, SVM classifier is constructed for malware detection. The promising experimental results on real data collection demonstrate that our frame works out performs other alternate data mining base detection methods in identifying new malicious executables. Unlike the previous researches which are unable to mined is criminative features, we propose to use sequence mining algorithm on instruction sequence to extract well representative features.

8.2 Future Enhancement

Future work includes partitioning of the original gene set into some distinct subsets or clusters so that the malware within a cluster are tightly coupled with strong association to the sample categories. We can extend the work to implement various classification algorithms to improve the accuracy rate at the time of malware prediction

APPENDIX – I

SCREENSHOTS

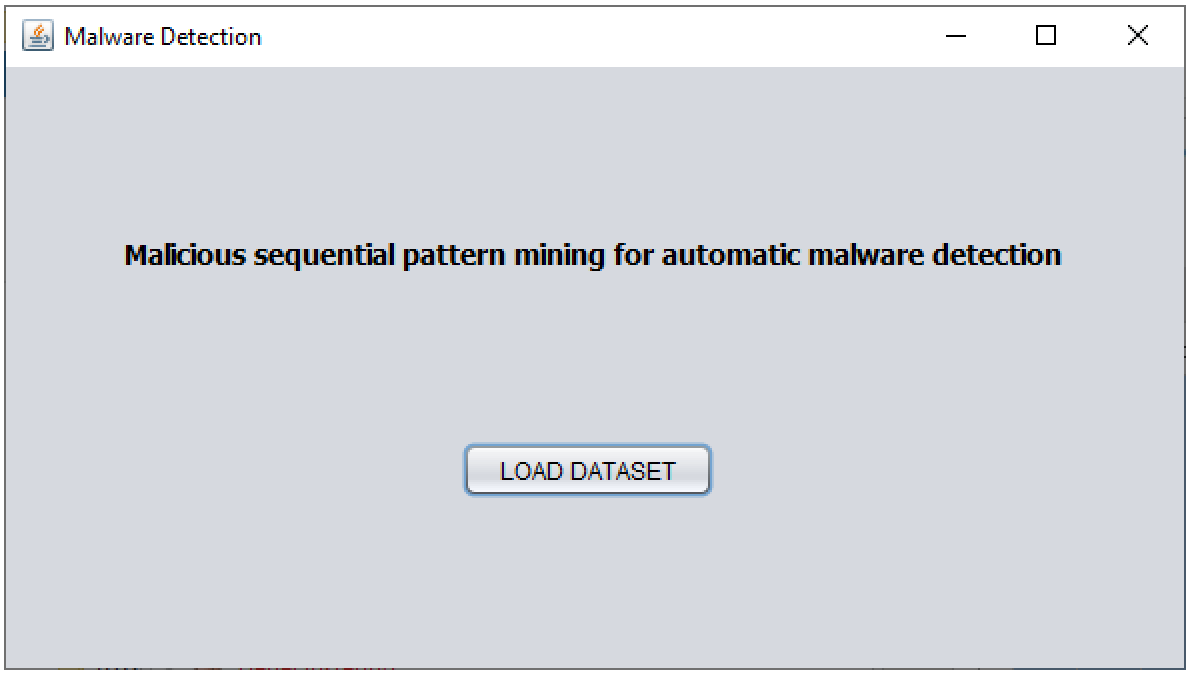
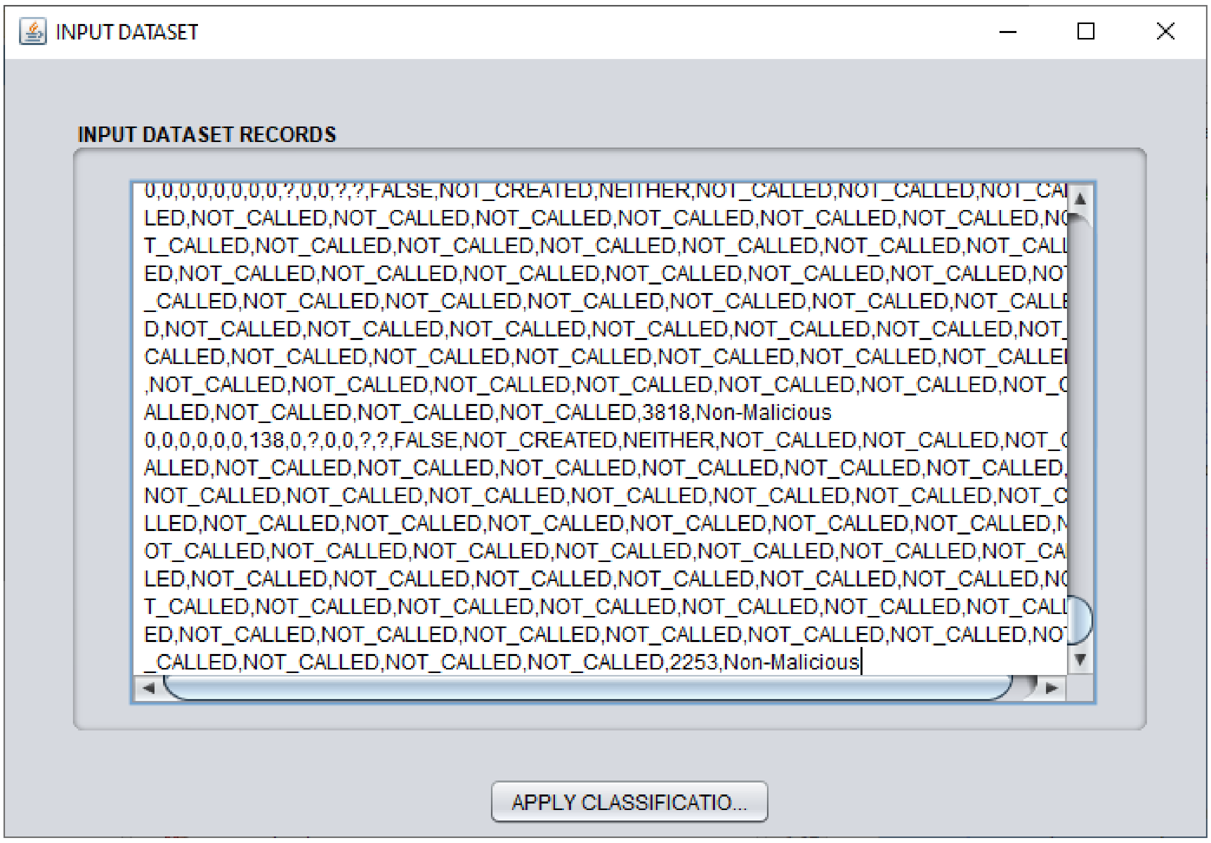
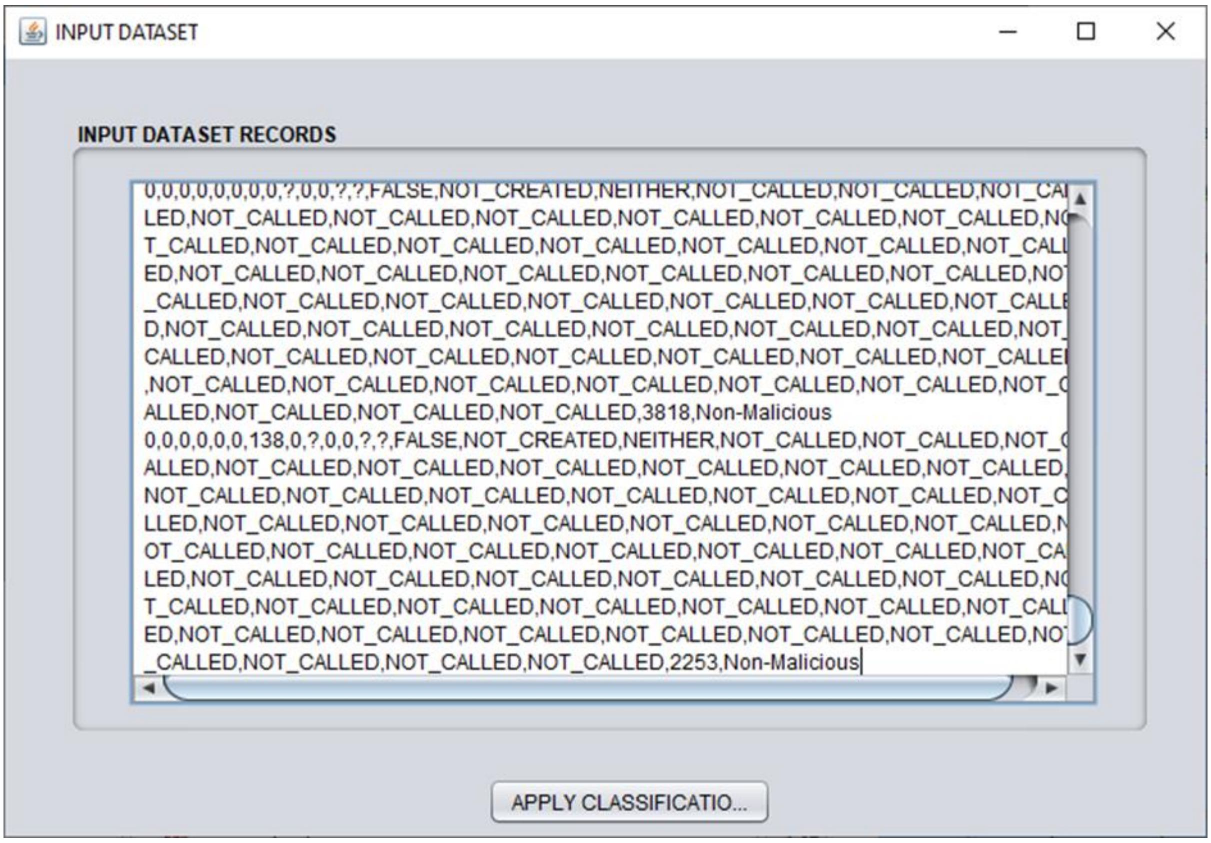

Malware Detection
Malicious sequential pattern mining for automatic malware detection
LOAD DATASET

Fig A.2.1 Loading Dataset



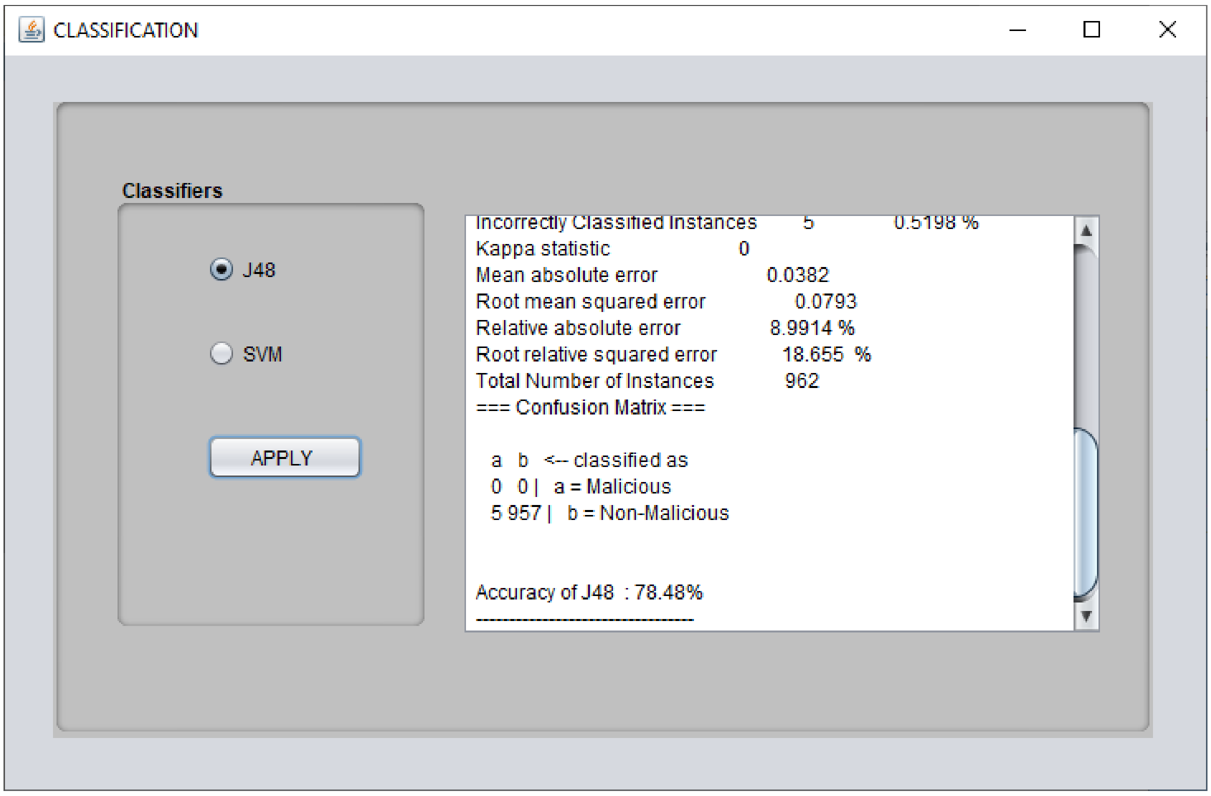
```
0,0,0,0,0,0,0,0,?,0,0,?,?,FALSE,NOT_CREATED,NEITHER,NOT_CALLED,NOT_CALLED,NOT_CA
LED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NO
T_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALL
ED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT
_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLE
D,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_
CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED
,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_C
ALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,3818,Non-Malicious
0,0,0,0,0,0,138,0,?,0,0,?,?,FALSE,NOT_CREATED,NEITHER,NOT_CALLED,NOT_CALLED,NOT_C
ALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,
NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_C
LLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,N
OT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CA
LED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NO
T_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALL
ED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT
_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,2253,Non-Malicious
```

Fig A.2.2 Input Dataset



```
0,0,0,0,0,0,0,0,?,0,0,?,?,FALSE,NOT_CREATED,NEITHER,NOT_CALLED,NOT_CALLED,NOT_CAL
LED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NO
T_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALL
ED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT
_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLE
D,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_
CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLE
,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_C
ALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,3818,Non-Malicious
0,0,0,0,0,0,138,0,?,0,0,?,?,FALSE,NOT_CREATED,NEITHER,NOT_CALLED,NOT_CALLED,NOT_C
ALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,
NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_C
LLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,N
OT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CA
LED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NO
T_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALL
ED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,NOT
_CALLED,NOT_CALLED,NOT_CALLED,NOT_CALLED,2253,Non-Malicious
```

Fig A.2.3 Training Dataset



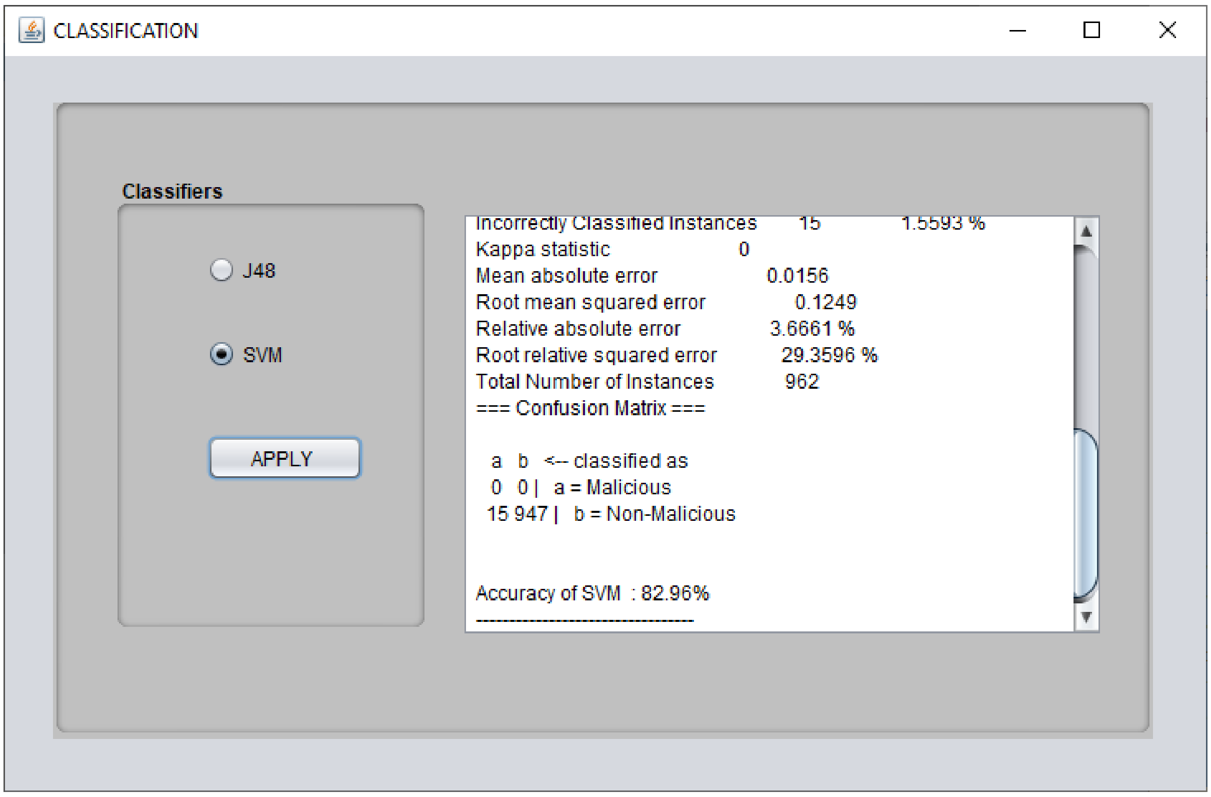
(● J48) ○ SVM

```
Incorrectly Classified Instances       5              0.5198 %
Kappa statistic                    0
Mean absolute error                  0.0382
Root mean squared error                0.0793
Relative absolute error              8.9914 %
Root relative squared error            18.655  %
Total Number of Instances             962
=== Confusion Matrix ===

   a   b   <-- classified as
   0   0 |   a = Malicious
   5 957 |   b = Non-Malicious


Accuracy of J48  : 78.48%
----------------------------------
```

Fig A.2.4 J48 in Classifier



```
Incorrectly Classified Instances      15           1.5593 %
Kappa statistic                  0
Mean absolute error              0.0156
Root mean squared error            0.1249
Relative absolute error           3.6661 %
Root relative squared error        29.3596 %
Total Number of Instances          962
=== Confusion Matrix ===

  a   b   <-- classified as
  0   0 |   a = Malicious
 15 947 |   b = Non-Malicious


Accuracy of SVM  : 82.96%
------------------------------------
```

Fig A.2.5 SVM in Classifier

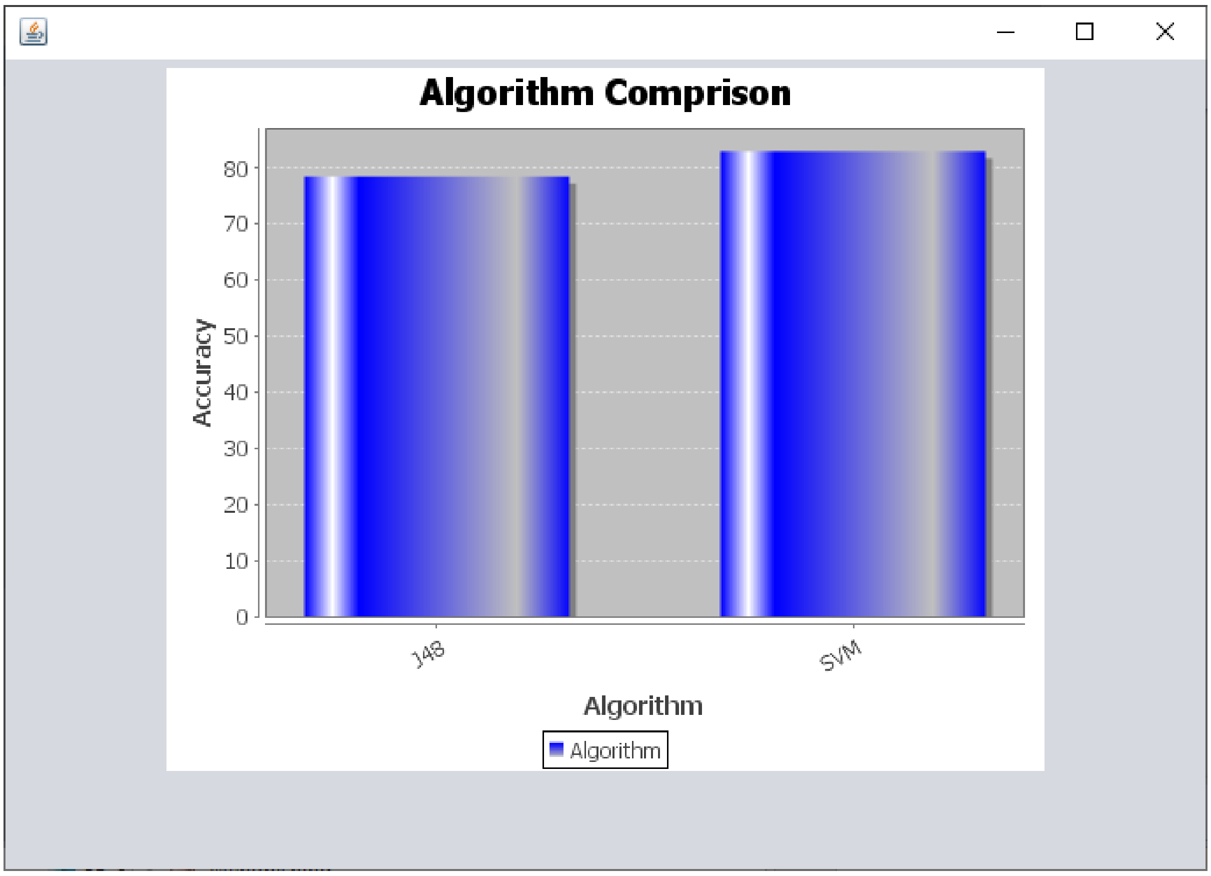


Fig A.2.6 Algorithm Comparison

REFERENCES:

[1] Ahmadi,M. ,Sami,A., Rahimi,H., & Yadegari,B, "Malware detection by behavioural sequential patterns", 2013.

[2] Akhtar, M.S.; Feng, T, "Detection of sleep paralysis by using IoT based device and its relationship between sleep paralysis and sleep quality", 2022.

[3] Gibert, D.; Mateu, C.; Planes, J.; Vicens, R, "Using convolutional neural networks for classification of malware represented as images", 2019.

[4] Narouei.M., Ahmadi.M., Giacinto.G., Takabi.H., & Sami,A., "DLL Miner: Structural mining for malware detection", 2015.

[5] Nissim,N. , Moskovitch,R. , Rokach,L. , & Elovici,Y, "A Noval approach for active learning methods for enhanced PC malware detection in windows OS", 2014.

[6] Rad,B. B.,Masrom, M. ,& Ibrahim,S, "Opcodes histogram for classifying metamorphic portable executable small ware", 2012.

[7] Wchner,T. , Ochoa,M. , & Pretschner,A, "Malware detection with quantita-tive data flow graphs", 2014.

[8] Ye,Y., Li, T.,Chen, Y., & Jiang, Q, "Automatic malware categorization using cluster ensemble", 2010.

[9] Ye, Y., Wang, D., Li, T., Ye, D., & Jiang, Q , "An intelligent PE-malware detection system based on association mining", 2008.

[10] Zhao, K.; Zhang, D.; Su, X.; Li, W, "Fest: A feature extraction and selection tool for android malware detection", 2015.