DYNAMIC ANALYSIS OF MALICIOUS CODE IN A NON-SIMULATED ENVIRONMENT

BY

BRUNO R. NADER

A Thesis Submitted to the School of Graduate Studies
In Partial Fulfillment of the Requirements for the Degree of
Master of Science

Southern Connecticut State University
New Haven, Connecticut
August 2014

DYNAMIC ANALYSIS OF MALICIOUS CODE IN A NON-SIMULATED ENVIRONMENT

BY

BRUNO R. NADER

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Hrvoje Podnar, Department of Computer Science, and it has been approved by the members of the candidate's thesis committee. It was submitted to the School of Graduate Studies and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

Hrvoje Podnar, Ph.D.
Thesis Advisor

Amal Abd El-Raouf, Ph.D.
Second Reader

Lisa Lancor, Ph.D.
Department Chairperson

ABSTRACT

Author: Bruno R. Nader

Title: DYNAMIC ANALYSIS OF MALICIOUS CODE IN A NON-SIMULATED ENVIRONMENT

Thesis Advisor: Dr. Hrvoje Podnar, Ph. D.

Institution: Southern Connecticut State University

Year: 2014

This thesis presents a dedicated system to analyze any binary file, and classify it as malicious or benign. Behavioral data was collected over a period of time and used to determine the appropriate code category.

Unknown code was categorized using statistical measures such as standard deviation, mean and variance. Informative reports that included detailed data about the modifications made to the Windows registry, the file systems, and any other unusual processed information were analyzed for this purpose.

The experiments were conducted through the use of code specifically written for the task along with several commonly used software packages for the purpose of hard drive formatting and data manipulation.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Dr. Podnar for all of his help with this project, for all of the nights that we spent together at his office talking about this project, and other technologies and facts within the Computer Science world. I also would like to thank Dr. Abd El-Raouf for being my second reader on this project, and for supporting my thesis work. I really learned a lot from her during classes, which gave me a huge interest in this project and various other topics. I also would like to thanks Dr. Syed for all the classes that I had with her, for all the material that was well presented to us, and for making us learn and think about the problems in a different way. I really enjoyed her classes and learned a lot with her.

Finally, I would like to thank my wife, Claudia Ferreira, for helping me with this project by giving me support, and for taking care of our son, Brendon, during my long nights at Dr. Podnar's office. I also would like to thank my son, Brendon Nader, and my daughter, Barbara Nader, for their support and understanding. And, lastly, I would like to thank my Mom, Maria Do Socorro Vieira Nader, for being a wonderful mother, for always encouraging me to continue studying, and for the education that she provided me with. Thank you, Mom, for being a great mother, and for helping to make me the man I am today. I love you all, Claudia, Brendon, Barbara, Mom, and Dad!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

This chapter serves as an introduction to the different types of computer system vulnerabilities, and the description of the malicious code detection problem. This chapter will also present the different detection approaches used to detect and classify an unknown binary code. This chapter will conclude with the statement of purpose presented for this thesis.

1.1 Types of Computer Systems' Vulnerabilities

With the popularity of the Internet, the presence of malicious code has become widespread. Not only has it increased in volume, but also in complexity and sophistication. Malicious code has exhibited abilities to avoid detection and/or deletion [1]. This has presented a serious risk to the security of computer systems, threatening the integrity and confidentiality of personal information [2]. Currently malicious code categories are associated to types of activities performed. These categories include:

- Viruses
- Trojans
- Adware
- Worms

According to the Microsoft Security Intelligence report, released in July of 2012, viruses are the most common and yield approximately 57% of all malicious code [3]. A virus is characterized as a code that can replicate itself without end user interaction [4]. Viruses can damage file systems and/or take over the use of system resources. Typically, a virus might attach itself to a legitimate host application program, and then complete execution when the code is activated [5].

A Trojan, on the other hand, is generally a self-contained malicious program that will not self-replicate but will perform malicious actions on users' computers without their knowledge. Users need to download Trojans [6]. Therefore, Trojans need to disguise themselves as fully functional programs with desirable capabilities. When Trojans are triggered, by being downloaded onto a computer, they can cause loss, theft of data, or create back doors to the users' computer. This will allow hackers access to the system, allowing confidential and personal information to be compromised [7].

Another example of malicious code is Adware. Adware is a software package that shows advertisements to end-users in order to generate revenue for the author of the adware. Usually the advertisements are inserted within the display of other software. When the user downloads the software, the ads are attached to the software [8]. Adware is usually installed prior to the desired software and/or tool has completed being downloaded. In this case, the Adware will be disguised within the software [9]. Then, when we are prompted to click finish and say yes to complete the install, the user executes the Adware unintentionally in order to acquire the desired software and/or tool. While running the legitimate software, the user will be exposed to undesirable ads. These ads may be presented to the user many times, in the form of popup screens, characteristic of their bothersome and potentially malicious nature [10].

A worm is a standalone program that uses the network to replicate itself in order to spread to other computers through local network connections. A worm differs from a virus in that it does not need to attach itself to a host application in order to replicate. A worm harms the network by consuming bandwidth and slowing down the rest of the machines in the network [11].

1.2 Detection Approaches

There are two main approaches for detection of malware: static and dynamic analysis. Static analysis is the actual viewing and parsing through the code, and can be done when scanning the binary code with anti-virus software [12]. First, one would look at the malware with a hex editor, then unpack the malware, and then perform string searches and disassemble the malware. Dynamic analysis is a way to understand how the malware behaves when executed, what it talks to, what gets installed, and how it runs [13]. Dynamic analysis also consists of monitoring the execution of the program to detect malicious behavior [14].

Static analysis has the advantage of being able to analyze the complete code. It is usually faster than dynamic analysis and, as such, can be used against multiple executables in a short period of time. By using the static analysis approach, virus scanners (a database of descriptions or signatures that characterize known malware instances) are unable to detect malicious code that changes itself over a period of time unless a new signature is defined for the mutated code [15].

In contrast, dynamic analysis utilizes indirect measures to follow the state of a computer system. Such measures are used to detect the malicious features of the running code. Some measures involve monitoring changes in the file system, registry key values, processes, and services list [15]. Dynamic analysis has also been tested by a number of malware detection systems. Some of those systems include CWSandbox [16], Norman Sandbox [17], TTAnalyze [18], and Cobra [19]. The sandbox approach requires automatically loading the malicious code into a virtual machine environment and executing it. This approach records system calls, initiates processes and Windows registry changes during the execution of the code, and generate reports about the activity [20].

Despite the facts that the dynamic analysis approach reduces the workload of the human analyst, it has drawbacks. Malicious code is often equipped with detection routines that check for the presence of a virtual machine or a simulated OS environment [11]. When malware detects such an environment, it will modify its behavior and not perform malicious actions producing incorrect results. In this case, the code will be classified as benign. Another drawback of this dynamic approach is that only one piece of malicious code can be analyzed during a test. This is done in order to reduce interference between multiple applications and to isolate the behavior of the code in question [21].

1.3 Statement of Purpose

This thesis presents a new way to analyze, detect, and to qualify an unknown piece of possible malicious code. It also describes an unconventional way of running the questionable code in a real system instead of a virtual machine. We will now present a new approach of detecting, analyzing, and categorizing suspicious code by using the results of running the code on an actual system. Other signature-based detection systems compare the contents of the file to be tested against known malicious code.

This research project utilizes a snapshot from the system properties before and after the suspect code was run on the system. The system snapshot includes all the registry key entries and a list of the files present in the system. In this approach, we are not able to monitor the activity of malicious code since the analysis is done long after the code has done its damaging tasks to the system.

In this thesis, an approach is presented that will attempt to detect the level of malicious behavior after the attack has occurred. In order to accomplish this task, a computer system was created with multiple hard drive partitions containing separate, but identical, Windows XP Professional operating system installations. This was done, instead of installing the system on a virtual machine, in order to counteract the ability of the virtual machine, or simulated operating system, detection built into many types of malicious code. This type of detection would then enable the malicious code to escape detection by staying dormant, or altering its behavior, resulting in the inability to create a signature used to detect a malicious piece of software.

For this project, we are able to provide the security community with other dynamic analysis tools. This system is able to detect and classify any suspicious code as being malicious or benign with a 90% success rate.

CHAPTER 2: SYSTEM DESIGN

This chapter describes the high-level system architecture necessary to build the Malicious Code Detection (MaCod) system. It will also explain the steps, and the different parts of the system, in order to allow for the unknown binary code to run. How the system can then quickly be brought back to its clean and initial state for the analysis of another binary code will also be addressed.

2.1 High Level System Architecture

Usually users do not realize that their computer systems have been affected until after the malicious activity has been completed. The evidence of the malicious activity is present on the computer system hard disk (HD) drive. The approach used by the system that was designed for this thesis requires a snapshot of the computer system before and after the unknown activity has occurred.

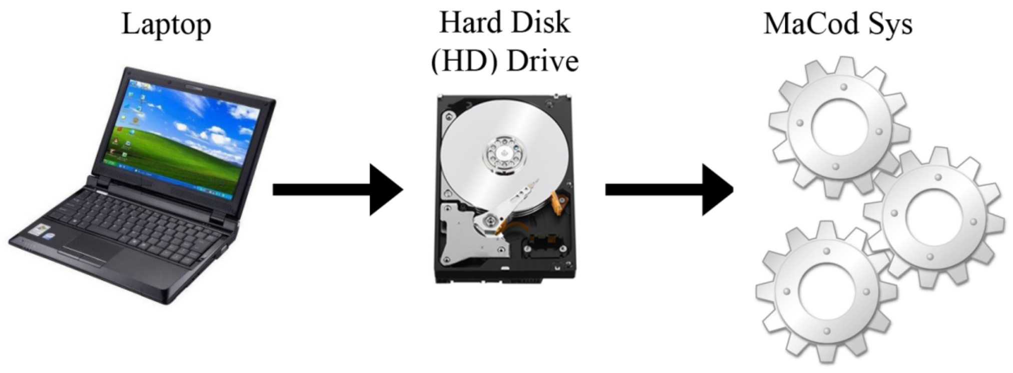


Figure 2.1.1 - Malicious Code Detection (MaCod) System.

These snapshots are then utilized for analysis and detection purposes. The MaCod system will use a number of analysis techniques in order to determine if the code that affected the MaCod system is malicious in nature. This is done in multiple stages, starting from a healthy system and possibly ending up with an affected system. To make this possible, the approach started with the same operating system and file structure configuration. The experiment was performed by utilizing unknown code run for a pre-specified length of time. The final state of the computer system was recorded and used for analysis. The complete process is done on a single workstation.

2.2 Hard Disk Partitioning

This section will explain how the system hard disk is divided and the reason behind its structure. In the MaCod system, the hard drive is partitioned into 3 parts with the following names:

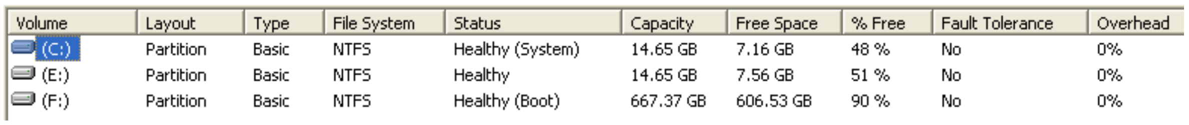
- The Experimental Partition

- The Clean Partition
- The Running Partition

The Experimental Partition, like its name describes, is used to run the unknown binary code. This partition is the sandbox of the system, where the binary code is allowed to perform all of its actions for a predefined amount of time. This partition is fairly small, containing 14.65 gigabytes (GB) of total space. The reason for creating such a small partition is to enable the system to be restored back to its initial and clean state fairly quickly.

The next partition of the system is the Clean Partition. This partition is similar to the Experimental Partition, containing the same size of 14.65 GB of space. As a matter of fact, the Experimental Partition is a cloned copy of the Clean Partition. This partition is necessary to allow the Experimental Partition to be restored back to its initial and clean state after an unknown binary code has finished running.

The Running Partition is used to store and generate the analysis reports during the data collection process. This partition is also used to store the source code for the custom software, called RegKeg, which was designed to help in the collection of the data and the generation of the analysis reports. All of these analysis reports are also stored in this partition for further analysis at a later time. This partition is set to the remaining size of the hard disk that, in the case of this thesis experiment, was set to 667.37 GB of total space, as it is shown in Figure 2.2.1 below:



Volume	Layout	Type	File System	Status	Capacity	Free Space	% Free	Fault Tolerance	Overhead
(C:)	Partition	Basic	NTFS	Healthy (System)	14.65 GB	7.16 GB	48 %	No	0%
(E:)	Partition	Basic	NTFS	Healthy	14.65 GB	7.56 GB	51 %	No	0%
(F:)	Partition	Basic	NTFS	Healthy (Boot)	667.37 GB	606.53 GB	90 %	No	0%

Figure 2.2.1 – MaCod’s Hard Disk Partitions

2.3 MaCod System Design

The main modules of the MaCod System are presented in Figure 2.3.1.

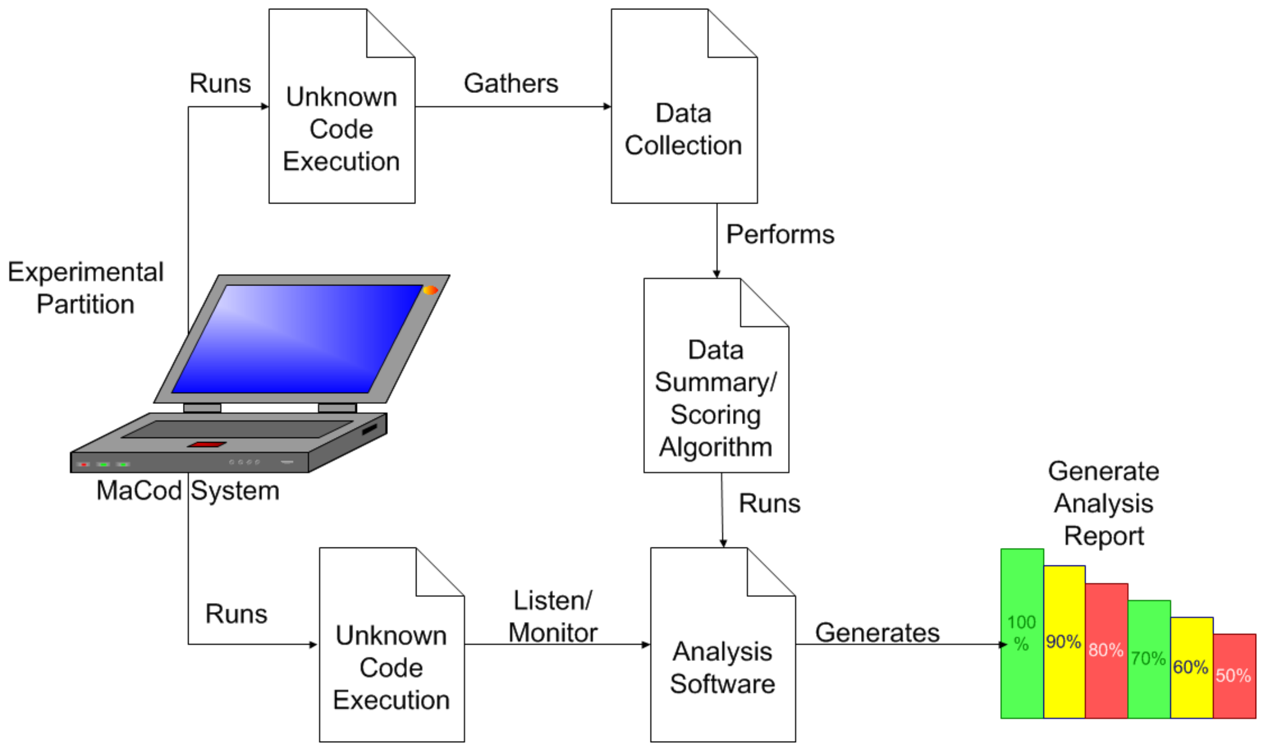
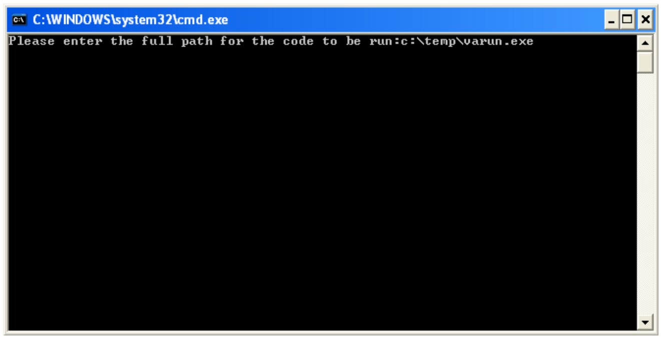


Figure 2.3.1 – MaCod System Design

The system has various stages in order to determine if the code in question is malicious or not. These stages are: known and unknown code execution, data collection, data summary, a scoring algorithm, and an analysis report with the final malicious classification results.

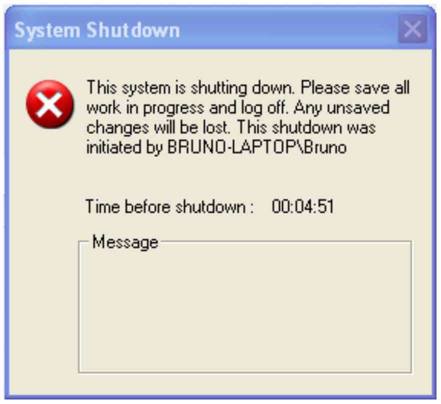
The first module to be run by the user is the module responsible for collecting the data for the system before any unknown code has been run. This will be the before dataset. After this process, the unknown binary code can then be run in the experimental partition. For this process, a module was created called the “Binary Code Runner”. This is a DOS command line program that will ask the user for the path of the unknown code to be run as shown in Figure 2.3.2 below:



```
Please enter the full path for the code to be run:c:\temp\varun.exe
```

Figure 2.3.2 – Binary Code Runner Module

After the user enters this information correctly, the system will run the code and will start a timer to halt and restart the system after 5 minutes (as shown in Figure 2.3.3 below).



This system is shutting down. Please save all work in progress and log off. Any unsaved changes will be lost. This shutdown was initiated by BRUNO-LAPTOP\Bruno

Time before shutdown : 00:04:51

Figure 2.3.3 – System Shutdown Timer

After the system is restarted into the running partition, the user will need to run another module that is responsible for collecting the data and generating the analysis reports. This after dataset will then be stored and saved for further analysis at a later time.

After collecting the after dataset, the user will then allow RegKeg to compare the before and after dataset by initiating another module (as shown in Figure 2.3.4). The user would click on the menu button that reads "Start Data Analysis and Reporting."

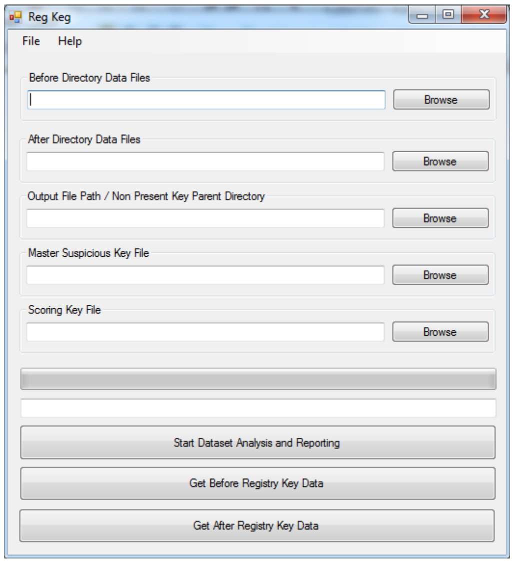
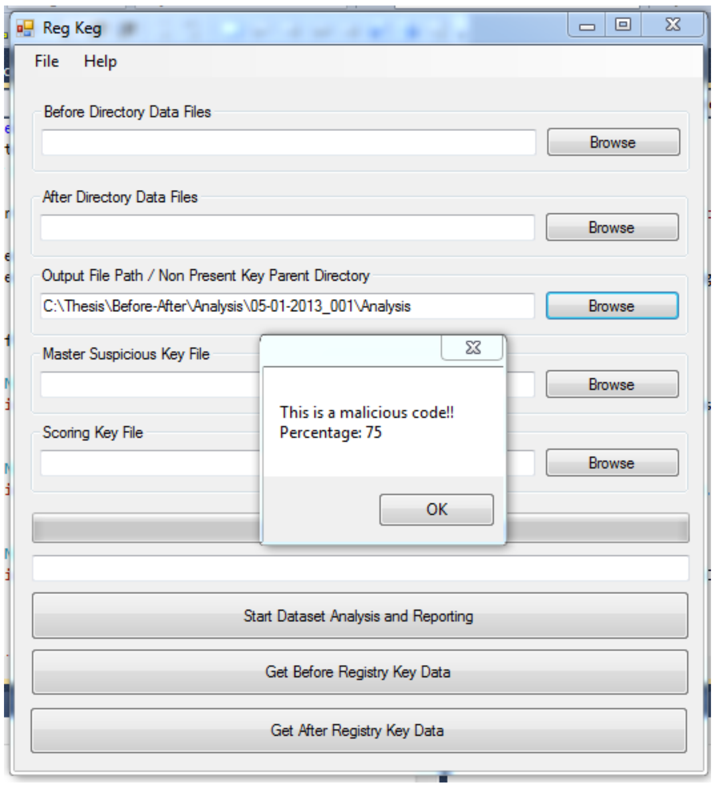

Reg Keg
File
Help
Before Directory Data Files
Browse
After Directory Data Files
Browse
Output File Path / Non Present Key Parent Directory
Browse
Master Suspicious Key File
Browse
Scoring Key File
Browse
Start Dataset Analysis and Reporting
Get Before Registry Key Data
Get After Registry Key Data

Figure 2.3.4 – RegKeg User Interface

After comparing the before and after datasets, RegKeg will generate statistical data for that unknown binary code. This statistical data will contain the average, the median, and the standard deviation of the score in respect to the entire dataset that was collected during the training process of the system.

After generating the statistical data, the user can then run the code classification module, which will read the statistical data that was generated in the previous step. This module will then determine the code classification of the unknown binary code based on the data that was collected during the training process. This module will then display to the user the results in the form of an alert message, as show in Figure 2.3.5 below.


Reg Keg
File
Help
Before Directory Data Files
Browse
After Directory Data Files
Browse
Output File Path / Non Present Key Parent Directory
Browse
Master Suspicious Key File
Browse
Scoring Key File
Browse
OK
Start Dataset Analysis and Reporting
Get Before Registry Key Data
Get After Registry Key Data

C:\Thesis\Before-After\Analysis\05-01-2013_001\Analysis

This is a malicious code!!
Percentage: 75

Figure 2.3.5 – Code Classification Alert Message Result

This section concludes with a high level design of the MaCod System architecture. By following these instructions and processes, the user is able to gather data, generate analysis reports, and attempt to classify the unknown binary code that was run in the system. Another important aspect is that this configuration will allow the user to quickly restore the system back to a clean state, which will enable the user to run more binary codes for analysis.

CHAPTER 3: METHODOLOGY

In this chapter, the modules of the system will be described as the gathering of the data, and the reporting and intelligence behind it. This chapter will present the processes and steps necessary to perform these tasks.

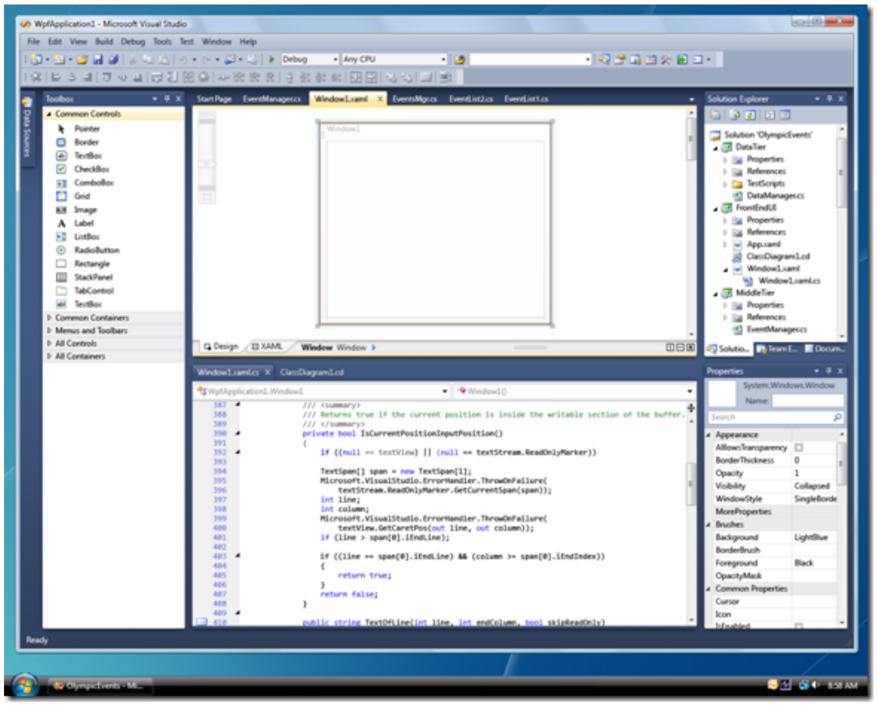
3.1 Tools Used to Build the MaCod System

This thesis utilized a series of software tools and custom code to build and test the experimental system. Below is a list of tools used for the system:

- Microsoft Visual Studio (VS) 2010 Integrated Development Environment (IDE)
- C# Programming Language/.NET Framework
- Gparted
- Clonezilla
- Windows Batch Scripting Language
- RegKeg – Custom Software

3.1.1 Visual Studio 2010 IDE

For this project, the decision was made to use VS 2010 Professional IDE for a variety of reasons. VS 2010 enabled the development of the project code in a timely manner due to the capabilities of efficient testing and debugging. The VS 2010 professional version is a free IDE for computer science students that provided an environment for software development as shown in Figure 3.1.1 below:



```
/// <summary>
/// Returns true if the current position is inside the writable section of the buffer.
/// </summary>
private bool IsCurrentPositionInputPosition()
{
    if ((null == textView) || (null == textStream.ReadOnlyMarker))

    TextSpan[] span = new TextSpan[1];
    Microsoft.VisualStudio.ErrorHandler.ThrowOnFailure(
        textStream.ReadOnlyMarker.GetCurrentSpan(span));
    int line;
    int column;
    Microsoft.VisualStudio.ErrorHandler.ThrowOnFailure(
        textView.GetCaretPos(out line, out column));
    if (line > span[0].iEndLine);

    if ((line == span[0].iEndLine) && (column >= span[0].iEndIndex))
    {
        return true;
    }
    return false;
}

public string TextOfLine(int line, int endColumn, bool skipReadOnly)
```

AllowsTransparency	☐
BorderThickness	0
Opacity	1
Visibility	Collapsed
WindowStyle	SingleBorde
Background	LightBlue
Foreground	Black

Figure 3.1.1 – Visual Studio 2010 Integrated Development Environment (IDE).

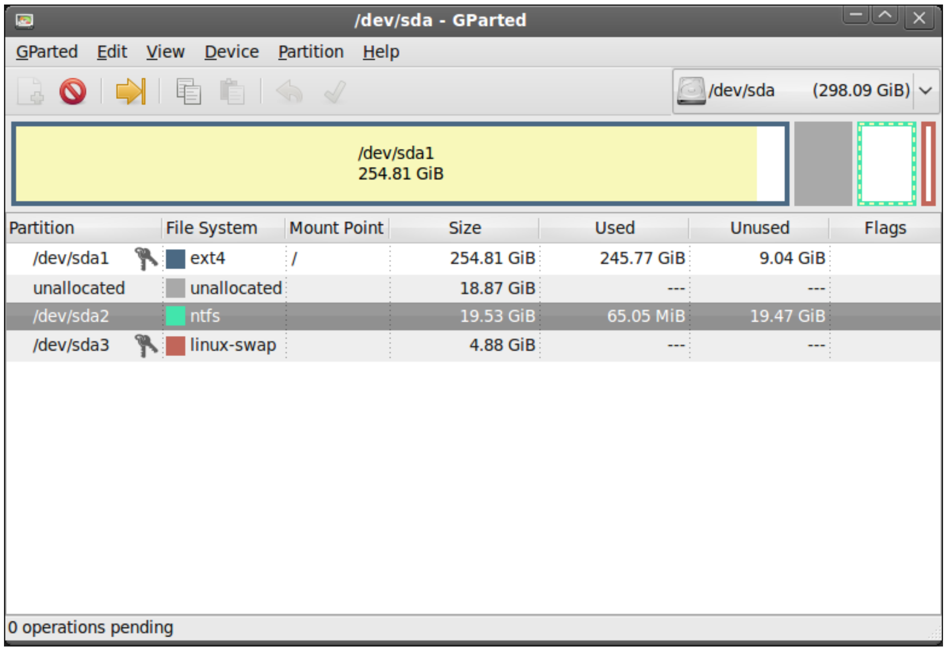
3.1.2 .NET Framework/C# Programming Language

The programming framework that comes bundled with the VS 2010 IDE is a software framework developed by Microsoft called the .NET (pronounced dot net) Framework. It includes a large library, and provides language interoperability across several programming languages [22]. C# .NET (pronounced c-sharp) was chosen from multiple languages supported by the .NET framework to create the custom code used in the experimentation. C# was designed to be a simple, general purpose, and object-oriented programming language [23].

3.1.3 Partitioning with Gparted

For the partitioning module, the system used Gnome Partition Editor, also known as Gparted, a disk partitioning software used to partition the hard drive. Gparted is a popular software package that is free, open source, portable, and user friendly with a seamless Graphical User Interface (GUI). The Gparted tool enables users to change partitions by sliding the partition size bars to set the appropriate partitions sizes. In addition, it supports the majority of file systems and formatting [24].

As seen in Figure 3.1.2 below, three partitions were created: NTFS, EXT4, and a Linux-swap file system.



Partition	File System	Mount Point	Size	Used	Unused	Flags
/dev/sda1	ext4	/	254.81 GiB	245.77 GiB	9.04 GiB	
unallocated	unallocated		18.87 GiB	---	---	
/dev/sda2	ntfs		19.53 GiB	65.05 MiB	19.47 GiB	
/dev/sda3	linux-swap		4.88 GiB	---	---	

0 operations pending

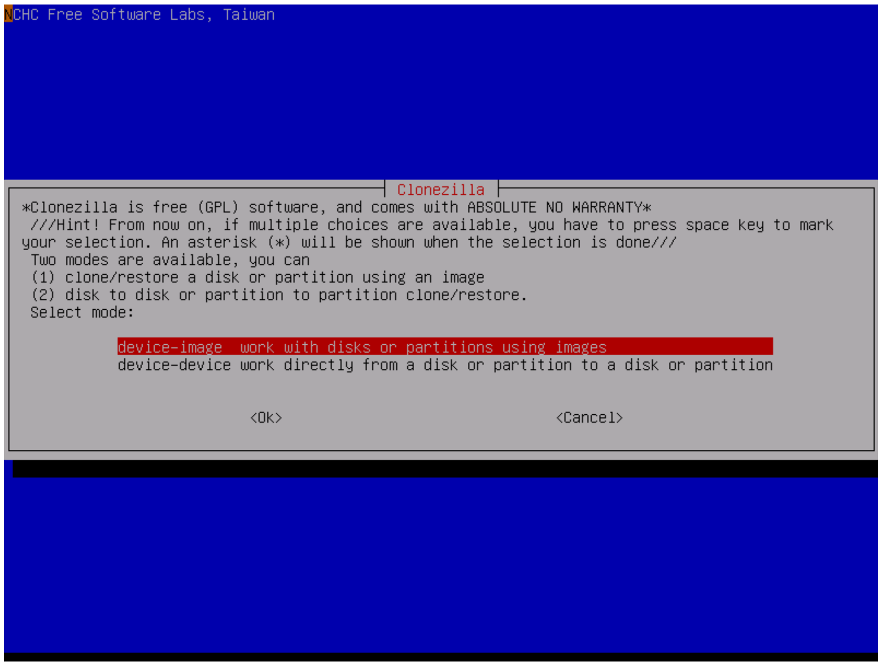
Figure 3.1.2 – Gnome Partition Editor (Gparted) Software.

When using the Gparted software, the hard drive of the system computer was split into 3 partitions. The Experimental and Clean partitions were both set to 14.6 gigabytes (GB), since the Experimental partition is a clone of the Clean partition. We chose a fairly small partition size for the Clean and Experimental partitions in order to speed up the process of imaging and cloning from one partition to another.

The Running partition was set to the remaining size of the hard drive that, in our case, was 667 GB. This partition was used to house data files and source code for the development of the custom software used to analyze and determine the degree of maliciousness of the file.

3.1.4 Cloning with Clonezilla

The other software tool that was used to build the system was Clonezilla, which is free and open source software for disk imaging and cloning [25]. Clonezilla was utilized before the testing to copy an image of a healthy system stored in the Clean partition onto the Experimental partition. The cloning is done by copying sector by sector from the original image to the Experimental partition from the Clean partition as shown in Figure 3.1.3. This process insures an exact clone of the Experimental partition.



```
NCHC Free Software Labs, Taiwan

Clonezilla
*Clonezilla is free (GPL) software, and comes with ABSOLUTE NO WARRANTY*
 ///Hint! From now on, if multiple choices are available, you have to press space key to mark
your selection. An asterisk (*) will be shown when the selection is done///
 Two modes are available, you can
 (1) clone/restore a disk or partition using an image
 (2) disk to disk or partition to partition clone/restore.
 Select mode:

             device-image  work with disks or partitions using images
             device-device work directly from a disk or partition to a disk or partition

                        <Ok>                                   <Cancel>
```

Figure 3.1.3 – Clonezilla Device to Device Cloning Mode

Figure 3.1.3 shows the menu selection that the user is presented with during the cloning process. In addition to cloning from partition to partition, Clonezilla provides options to clone to or from a disc image file. Clonezilla can be started from the command line by specifying the necessary parameters [25]. For example, the following command:

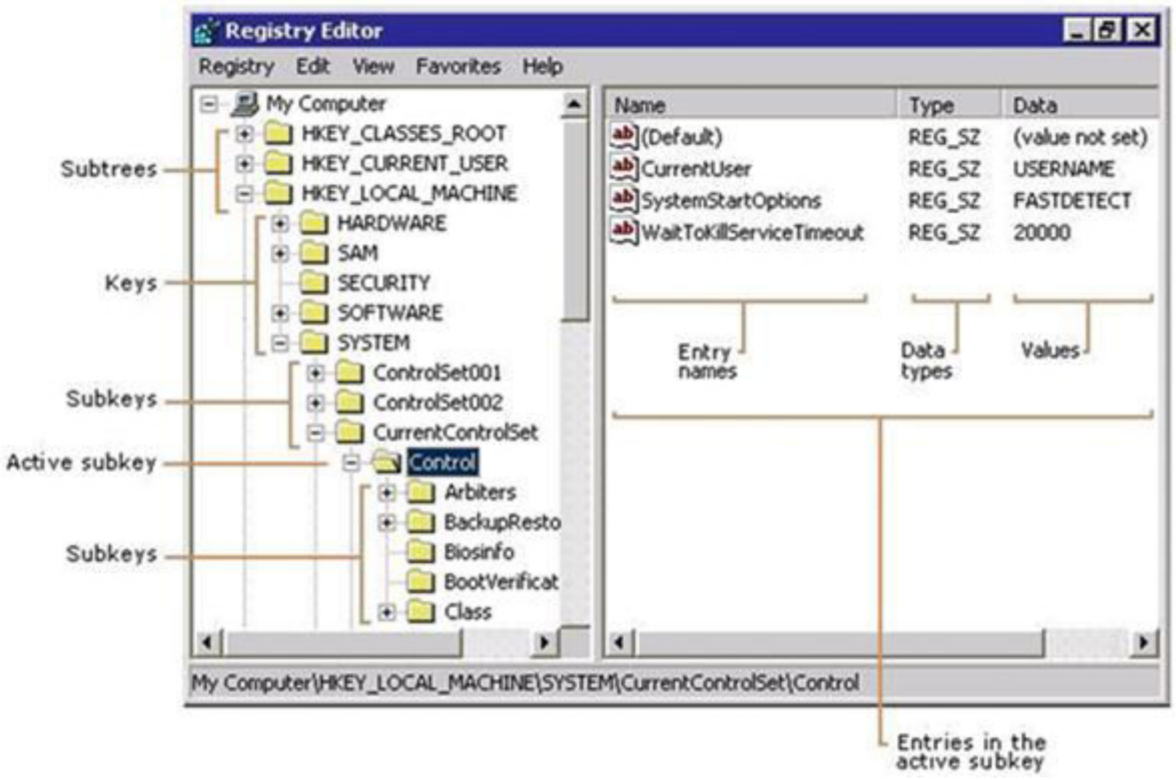
sudo /opt/drb1/sbin/ocs-onthefly –e1 auto –k –f sda1 –t sda2

The above command will create a clone copy of the sda1 (Clean) partition to the sda2 (Experimental) partition, with auto resizing as an additional feature of the program.

3.1.5 Before and After Tool

One of the tools used for data collection is the before and after tool. This tool's main purpose is to grab a snapshot of the complete directory structure, file listing, and system registry keys before and after suspect code has been executed.

The significance of the Windows Registry is that it stores configuration settings and options on Microsoft Windows Operating Systems [26]. In the process of installing any software, Windows Registry Keys will be generated and stored. Those keys are used by the operating system, the installed software, and any other software that needs to interact with the installed software. Applications use Windows Registry keys and/or files to store the various features generated by the program according to their necessity. This information might include licensing, recent actions, and settings as shown in Figure 3.1.4 below:



Name	Type	Data
(Default)	REG_SZ	(value not set)
CurrentUser	REG_SZ	USERNAME
SystemStartOptions	REG_SZ	FASTDETECT
WaitToKillServiceTimeout	REG_SZ	20000

```
My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control
```

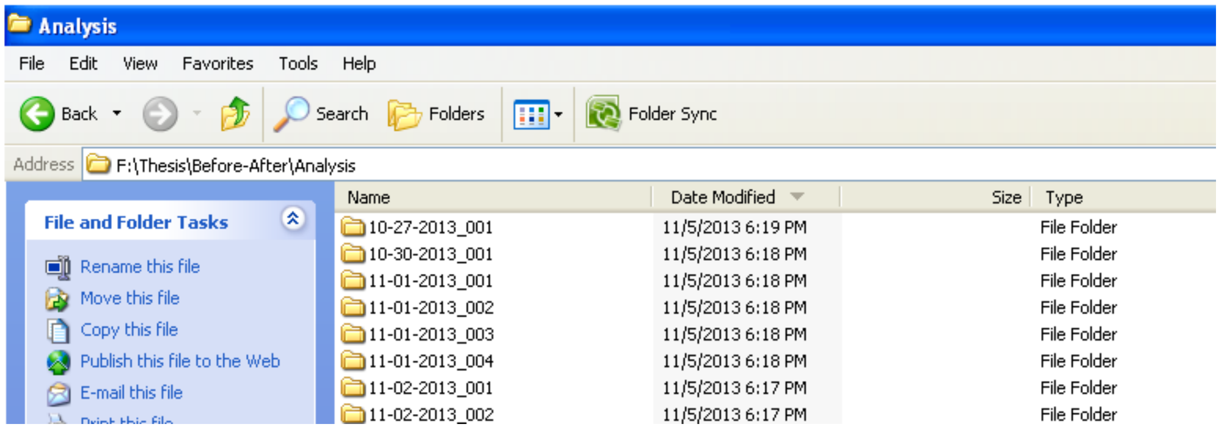
Figure 3.1.4 - Windows Registry

A batch file was created in order to perform this task. Written using the DOS command line and scripting language, the code enumerated through the entire operating system. It then generate a log of all the files, the directory structure, and all of the registry keys present in the Windows operating system at that time to be saved to the Running partition. This step was necessary for the analysis and comparison of both datasets when trying to determine the classification of the unknown binary code.

The batch file is called "snapshot.bat." This file was used twice, before and after the possible malicious code ran. If a system update, or a new software tool, needs to be installed this will be done and a new Clean partition image will be created. If a new Clean partition image is created, then we have to also grab a new before dataset by running the snapshot script against the

Clean partition. The snapshot script always runs in the Running partition, but the code will access different partitions depending when we are running the code. If we are running it to grab the before dataset, the code will then point to the Clean partition. If we are running it to grab the after dataset, the code will point to the Experimental partition.

All of the dataset sets that were captured by the snapshot script were saved for analysis and comparison by our custom software. A specific and easy to understand naming convention was adopted for the system. When saving the datasets, the snapshot script created directories using the following naming convention, which is in the form of "mm_dd_yyyy_xxx" that represents a date followed by the number of occurrences. An example of this would be 04_02_2014_001, which corresponds to April 2nd, 2014 for the 1st time in that day, as it is shown in Figure 3.1.5 below:



Name	Date Modified	Size	Type
10-27-2013_001	11/5/2013 6:19 PM		File Folder
10-30-2013_001	11/5/2013 6:18 PM		File Folder
11-01-2013_001	11/5/2013 6:18 PM		File Folder
11-01-2013_002	11/5/2013 6:18 PM		File Folder
11-01-2013_003	11/5/2013 6:18 PM		File Folder
11-01-2013_004	11/5/2013 6:18 PM		File Folder
11-02-2013_001	11/5/2013 6:17 PM		File Folder
11-02-2013_002	11/5/2013 6:17 PM		File Folder

Figure 3.1.5 – Directory Naming Convention

3.1.6 RegKeg

In addition to the tools described above, this project required the development of custom software in order to collect the data during the training process, perform the data analysis, and produce reports on the status of the binary code to determine whether the tested software remained benign or became affected by the malicious code.

The training process is a process wherein well-known binary codes were run against the system to collect the data. During the training process, the data was stored for further analysis and comparison against the entire dataset. This analysis was later used to get the average, the median, and the standard deviation for the scores found in the datasets. This tool, named RegKeg, performs the following tasks:

- Create the Before Key Dataset
- Binary Code Runner
- Create the After Key Dataset
- Build the Suspicious Dataset File
- Build General Statistics Data Points
- Before/After Dataset Analysis
- Classification of Binary Code

This is a sequence that the user needs to follow in order to use RegKeg and to be able to classify the binary code as it is shown in Figure 3.1.6 below:

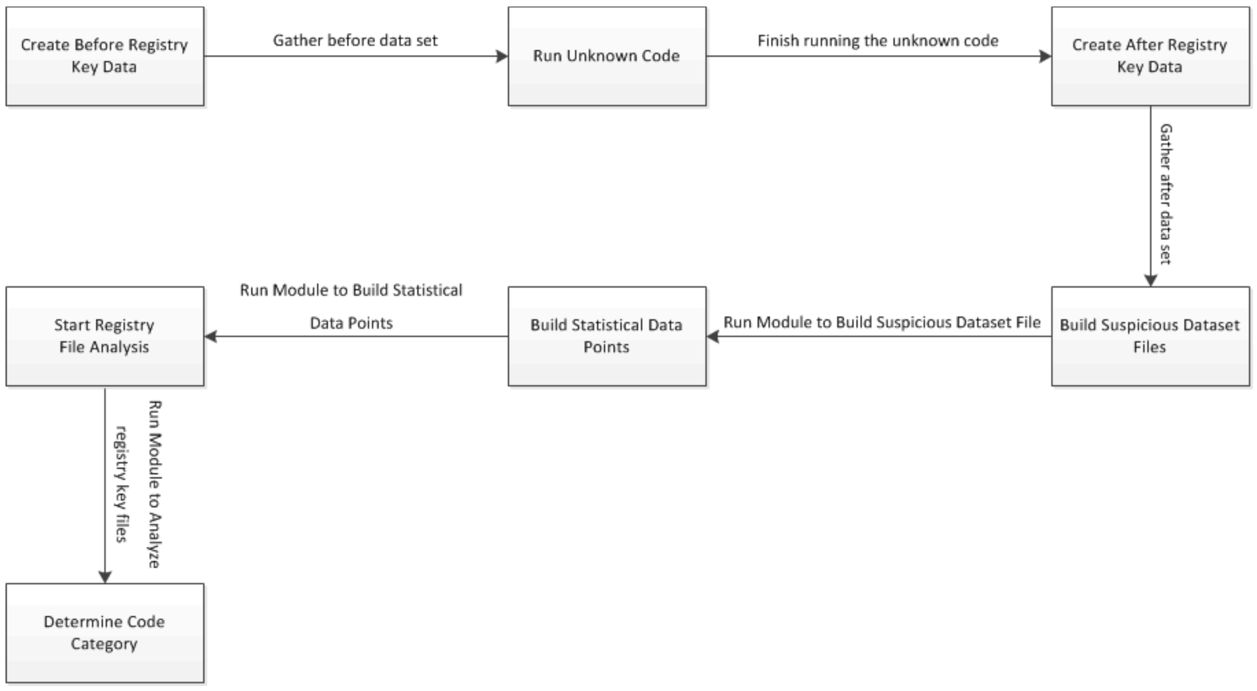

Create Before Registry Key Data
Gather before data set
Run Unknown Code
Finish running the unknown code
Create After Registry Key Data
Gather after data set
Build Suspicious Dataset Files
Run Module to Build Suspicious Dataset File
Build Statistical Data Points
Run Module to Build Statistical Data Points
Start Registry File Analysis
Run Module to Analyze registry key files
Determine Code Category

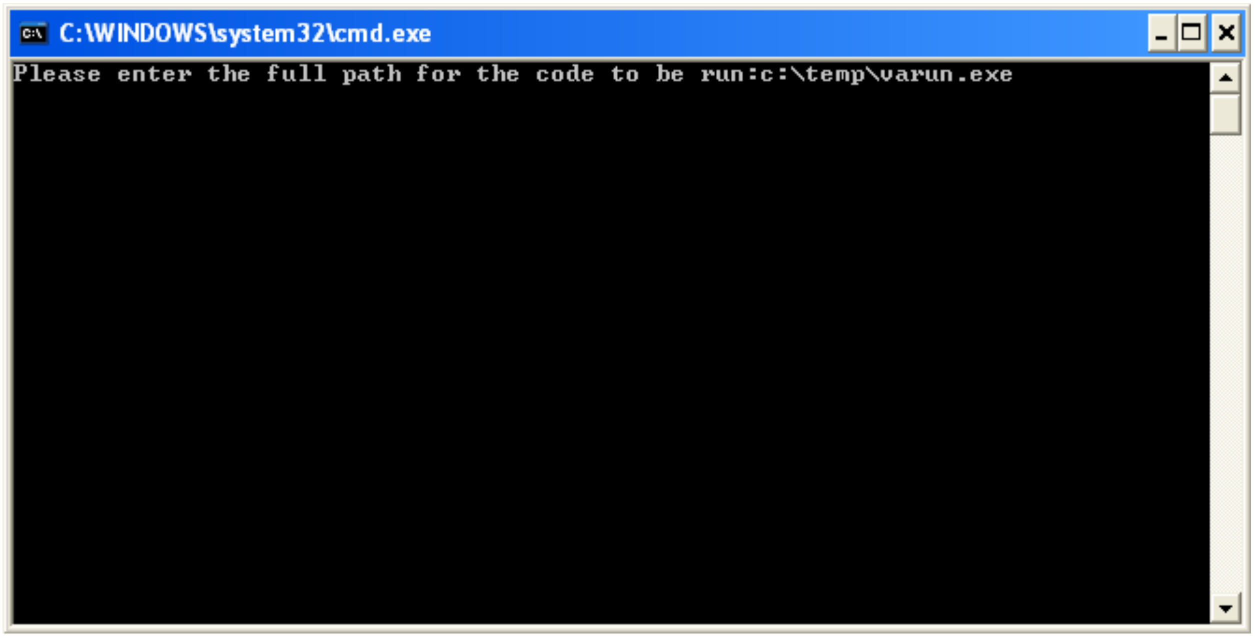
Figure 3.1.6 – RegKeg Workflow Diagram

3.1.6.1 Create the Before Key Dataset

This is the first operation to be performed when using RegKeg. This is done when the Clean partition image is established and is available to be cloned. When the user clicks on this menu item command, the RegKeg code will then call the snapshot.bat file. The scripting file will then enumerate through the system keeping a log of the entire file system and of all the registry key entries that are available for the Windows operating system. The batch file will also save the before dataset to the directory according to the naming convention that was described earlier in this chapter.

3.1.6.2 Binary Code Runner

The next operation is to restart the machine using the Experimental partition. As soon as the system is restarted, a command prompt window will appear and will ask the user to provide the path of where the binary code is present, in order to run the code, as shown in Figure 3.1.7 below:



```
Please enter the full path for the code to be run:c:\temp\varun.exe
```

Figure 3.1.7 – Binary Code Runner

The binary code runner will then run the unknown code, and will halt the system for five minutes, which is enough time for the unknown code to complete the changes to the MaCod System. This operation is performed to allow the binary code to perform its changes to the system and to allow analysis of the changes at a later time.

3.1.6.3 Create the After Key Dataset

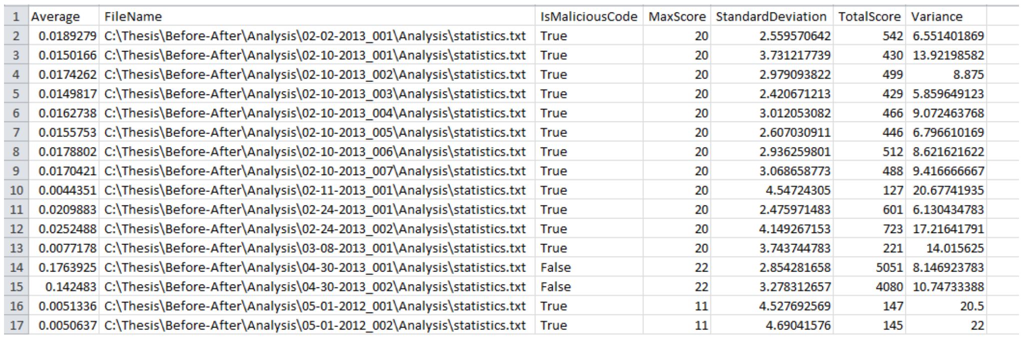
The action to create the after key dataset is performed by the snapshot script to grab the changes made by the unknown code, which will then call it the after dataset. This third module is run from the Running partition against the Experimental partition. The scripting file will enumerate through all of the system files and registry keys for the Windows system on the Experimental partition. This action will collect and log all of the files and entries for the system. Based on this collected data, RegKeg will compare the before and after dataset to detect any changes made to the system. This will later be used to generate the reports necessary for the software to classify the unknown binary code.

3.1.6.4 Build Suspicious Dataset File

The suspicious dataset is a list of all the files and registry keys that are found in the after dataset that were not present in the corresponding before dataset. All files and registry keys are compiled into a single suspicious dataset. This is done for the entire binary code sets that have ran in the experimental system. If the same registry key is found more than once, the weight for that key will be incremented by each time that it is found in the training data set.

3.1.6.5 Build General Statistics Data Points

This module will gather data points from all of the “statistics.txt” report files, described in the next section, that were generated during the analysis of each and every after dataset. This operation were performed to compile the statistics data points that were gathered during the training process into a single file, as it is show below in Figure 3.1.8:

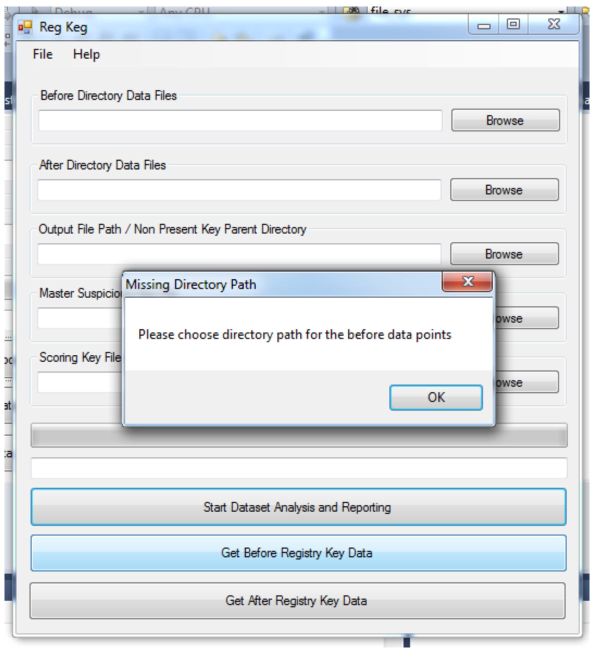


1	Average	FileName	IsMaliciousCode	MaxScore	StandardDeviation	TotalScore	Variance
2	0.0189279	C:\Thesis\Before-After\Analysis\02-02-2013_001\Analysis\statistics.txt	True	20	2.559570642	542	6.551401869
3	0.0150166	C:\Thesis\Before-After\Analysis\02-10-2013_001\Analysis\statistics.txt	True	20	3.731217739	430	13.92198582
4	0.0174262	C:\Thesis\Before-After\Analysis\02-10-2013_002\Analysis\statistics.txt	True	20	2.979093822	499	8.875
5	0.0149817	C:\Thesis\Before-After\Analysis\02-10-2013_003\Analysis\statistics.txt	True	20	2.420671213	429	5.859649123
6	0.0162738	C:\Thesis\Before-After\Analysis\02-10-2013_004\Analysis\statistics.txt	True	20	3.012053082	466	9.072463768
7	0.0155753	C:\Thesis\Before-After\Analysis\02-10-2013_005\Analysis\statistics.txt	True	20	2.607030911	446	6.796610169
8	0.0178802	C:\Thesis\Before-After\Analysis\02-10-2013_006\Analysis\statistics.txt	True	20	2.936259801	512	8.621621622
9	0.0170421	C:\Thesis\Before-After\Analysis\02-10-2013_007\Analysis\statistics.txt	True	20	3.068658773	488	9.416666667
10	0.0044351	C:\Thesis\Before-After\Analysis\02-11-2013_001\Analysis\statistics.txt	True	20	4.54724305	127	20.67741935
11	0.0209883	C:\Thesis\Before-After\Analysis\02-24-2013_001\Analysis\statistics.txt	True	20	2.475971483	601	6.130434783
12	0.0252488	C:\Thesis\Before-After\Analysis\02-24-2013_002\Analysis\statistics.txt	True	20	4.149267153	723	17.21641791
13	0.0077178	C:\Thesis\Before-After\Analysis\03-08-2013_001\Analysis\statistics.txt	True	20	3.743744783	221	14.015625
14	0.1763925	C:\Thesis\Before-After\Analysis\04-30-2013_001\Analysis\statistics.txt	False	22	2.854281658	5051	8.146923783
15	0.142483	C:\Thesis\Before-After\Analysis\04-30-2013_002\Analysis\statistics.txt	False	22	3.278312657	4080	10.74733388
16	0.0051336	C:\Thesis\Before-After\Analysis\05-01-2012_001\Analysis\statistics.txt	True	11	4.527692569	147	20.5
17	0.0050637	C:\Thesis\Before-After\Analysis\05-01-2012_002\Analysis\statistics.txt	True	11	4.69041576	145	22

Figure 3.1.8 – Statistics Report

3.1.6.6 Before and After Dataset Analysis and Reporting

When the user initiates this module, by clicking on the “Start Dataset Analysis and Reporting” button, five parameters need to be provided in order for the software to perform its tasks. If one of these necessary parameters is not provided, the user will receive an alert message and it will not proceed until the necessary parameter information is supplied as show in Figure 3.1.9 below:


Reg Keg
File
Help
Before Directory Data Files
Browse
After Directory Data Files
Browse
Output File Path / Non Present Key Parent Directory
Browse
Master Suspicio
Scoring Key File
Missing Directory Path
OK
Start Dataset Analysis and Reporting
Get Before Registry Key Data
Get After Registry Key Data

Please choose directory path for the before data points

Figure 3.1.9 – Parameter Message Alert

The necessary parameters that need to be supplied for this module to run are as follows:

- Path for the before dataset
- Path for the after dataset
- Path for outputting and saving analysis reports
- Path for the Master Suspicious Key dataset file
- Path for the Scoring Key dataset file

When the user supplies all five parameters, the module will start and it will perform several tasks:

- Read the before dataset

- Create the before registry key dictionary
- Read the after dataset
- Create the after registry key dictionary
- Create scoring dictionary
- Compare datasets
- Display comparison results

When reading the before dataset, the program will follow the path provided by the user and it will retrieve two files. These files are the “registry.reg” and the “FileSys.txt” files that are generated by the before batch file.

After retrieving these two files, the program will then create a before registry key dictionary. In computer science, a dictionary is a data structure that is capable of storing objects based on a unique key [27]. It is a key/value pair data structure that allows direct access to the object by providing the key. By using the dictionary data structure, RegKeg will quickly find out if a registry key is already part of the dictionary or not which allows the program to quickly retrieve the object by providing the registry key.

The same steps described above for the before dataset are used for the after dataset. After these steps are performed on the before and after datasets, the program will then have a dictionary containing a list of the before registry keys and another one containing the list of the after registry keys.

After this is complete, the program will then read the entire dataset from the “scoring_keys.txt” file, which is found in the fifth parameter that is provided by the user before running this module. The “scoring_keys.txt” file has 3 values separated by a “|” (pipe character) per line and this will be stored into a dictionary of keys and values. The first value is the registry key entry in the form of a string. The second value is the frequency that the registry key appears in the training dataset, represented by an integer. The third and last value is the weight that this key will have during the analysis and classification of the binary code.

The fourth task that happens in this module is to compare the before and after dictionary of registry keys. The way this is done is by going through each and every registry key in the before dictionary and doing a look up for the same key in the after registry key dictionary. It will then compare the values for this key. If the values are the same, nothing will happen but if they are different, for any reason, we will then log both values of the keys to a report file that later is used during the analysis process. When the comparison module is initiated, and the process occurs, the system uses various reporting and log files to help in the analysis process. Below is a list of these reporting and log files:

- Analysis.txt
- NonPresentKeys.txt
- Scoring_results.txt
- Statistics.txt

The "Analysis.txt" reporting file will log the value differences for the before and after registry key. The name of the registry key will be logged in one line, followed by the value for the before key and after key on separate lines. To separate the keys, and to make it easier to read and analyze, each entry is separated by dashes. Below is an example of what these entries look like:

```
[HKEY_LOCAL_MACHINE\reg6\Microsoft\DirectDraw\MostRecentApplication]

BEFORE:
"Name"="install_flashplayer11x32_mssd_aih.exe"
"ID"=dword:5010882b

AFTER:
"Name"="vlc-cache-gen.exe"
"ID"=dword:51f83c87
--------------------------------------------------------------------------------------------------
```

The other piece of data that is important for the analysis of the binary code, that is stored in the “NonPresentKeys.txt” file, is the registry key that is not found in the after dictionary of registry keys. Like the “Analysis.txt” file, this file follows the same criteria, which is key and value of the registry key followed by dashes as shown below:

```
Key: [HKEY_LOCAL_MACHINE\reg6\Classes\VLC.xm]
Value: @="VLC media file (.xm)"
-------------------------------------------------------
```

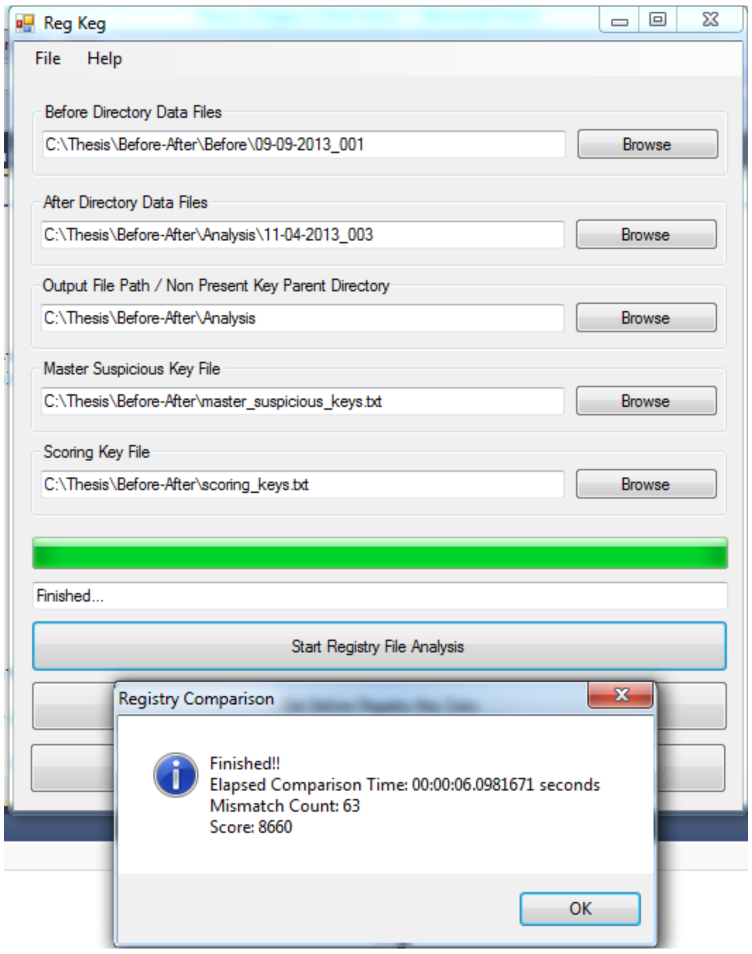
The next piece of data saved is what we call the scoring result dataset. This is saved into the “scoring_result.txt” file. For all the registry keys that are not found in the before dataset, but are contained in the scoring keys dictionary set, the registry key value will be separated by an equal sign followed by the weight for that key will be saved to the file. Here is an example:

```
[HKEY_LOCAL_MACHINE\reg2\Software\Microsoft\Windows\ShellNoRoam\BagMRU\0\2\1]=13
```

After all the keys have been read, the program will then total the scores for all the keys found in the scoring results dictionary. Then, it will take the average, median, and standard deviation for this dataset against all of the datasets that were collected throughout the experiment process.

After comparing the registry key datasets, the system will then compare the before and after file system data files. The way that this is accomplished is by reading the before and after file system files into the List of strings data type, and storing it in memory. A List in computer data structures is a collection of items where each item holds a relative position with respect to the others [28]. After the before and after list of file system entries are stored in the List objects, the system will perform a comparison and will report back with an entry to the "file_system_comparer.txt" file. The program will then state the name of the file that was not present in the before dataset, but is now present in the after dataset List file system.

After this process is accomplished, and all the reports are generated, the program will display the comparison results to the user in the form of an alert popup message. The message will state that the process has finished successfully, and it will also show the time that it took to run this module, the number of registry key mismatches, and the total score for all the keys that were found as show in Figure 3.1.10 below:



Before Directory Data Files: C:\Thesis\Before-After\Before\09-09-2013_001

After Directory Data Files: C:\Thesis\Before-After\Analysis\11-04-2013_003

Output File Path / Non Present Key Parent Directory: C:\Thesis\Before-After\Analysis

Master Suspicious Key File: C:\Thesis\Before-After\master_suspicious_keys.txt

Scoring Key File: C:\Thesis\Before-After\scoring_keys.txt

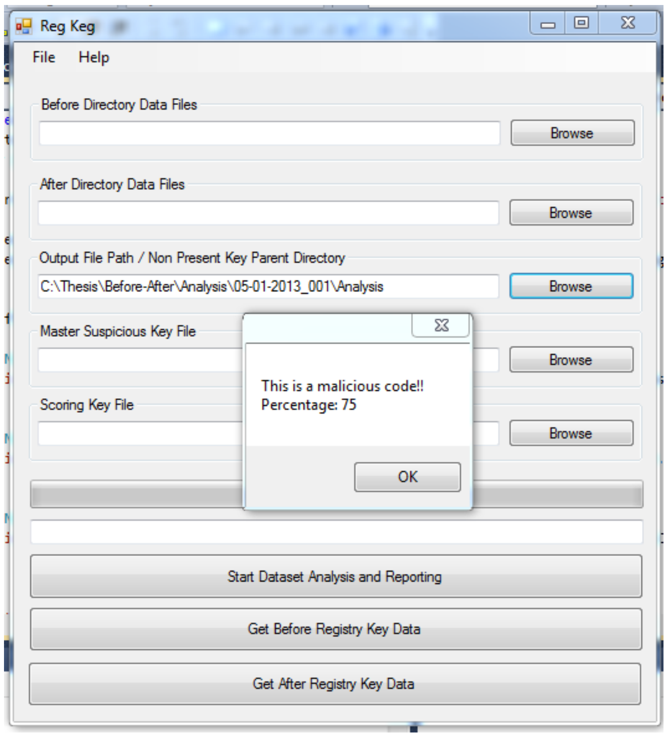
Finished...

```
Finished!!
Elapsed Comparison Time: 00:00:06.0981671 seconds
Mismatch Count: 63
Score: 8660
```

Figure 3.1.10 – Registry Comparison Results Message

3.1.6.7 Classification of Binary Code

After gathering all of the analysis reports and the scoring results data, the code may be classified. In this module, the program will attempt to classify the binary code as being benign or malicious. In order for this module to run, the user will need to provide one parameter, which is the statistics.txt file for that binary. This file was generated in the previous step, as we performed the before and after dataset analysis. After the system reads this file, it will then compare the values to specific threshold values that the system learned during the training process and the collection of the entire dataset that was executed during this experiment. During the training and analysis process, we have learned that a malicious code will usually have an average score greater than 0.02 and a standard deviation greater than 3.6 out of the total score of all the datasets that were studied and analyzed in the system. If this is the case, the program will classify the binary code as being malicious. If this is not the case, the program will classify it as being benign, which will be reflected in the form of a popup alert message as shown in Figure 3.1.11:


Reg Keg
File
Help
Before Directory Data Files
Browse
After Directory Data Files
Browse
Output File Path / Non Present Key Parent Directory
Browse
Master Suspicious Key File
Browse
Scoring Key File
Browse
OK
Start Dataset Analysis and Reporting
Get Before Registry Key Data
Get After Registry Key Data

C:\Thesis\Before-After\Analysis\05-01-2013_001\Analysis

This is a malicious code!!
Percentage: 75

Figure 3.1.11 – Binary Code Classification Alert Message

With this methodology in place, the system is able to identify and classify the type of binary code that was run in the system. This will help during the research process of a security analyst when trying to find out the results of what the binary code performed in the system.

CHAPTER 4: SYSTEM IMPLEMENTATION

This chapter describes the MaCod low-level system implementation details. It will describe the implementation of the before-after DOS batch script, the naming convention used to create the directories where the datasets are saved, and the implementation details of the RegKeg program and how it was used to gather and analyze the data necessary for this project.

4.1 Before and After Batch Script

To help with the process of gathering the dataset before and after running the unknown code in the Experimental Partition, we developed a Windows batch script named before-after.bat to automate the process. These scripts are responsible for taking a snapshot of all the Windows Registry keys and files in the system before and after we run the suspicious code. Below is a snippet of the before-after.bat batch file used by the MaCod system.

```
echo Loading Reg2 Hive - HKEY_CURRENT_USER KEY
reg.exe LOAD HKLM\reg2 "C:\Documents and Settings\Bruno\NTUSER.DAT"
echo Exporting Reg2 Hive - HKEY_CURRENT_USER KEY
reg.exe EXPORT HKLM\reg2 reg2.reg
echo.

echo Loading Reg6 Hive - HKEY_LOCAL_MACHINE - SOFTWARE KEY
```

```
reg.exe LOAD HKLM\reg6 "C:\WINDOWS\system32\config\software"
echo Exporting Reg6 Hive - HKEY_LOCAL_MACHINE - SOFTWARE KEY
reg.exe EXPORT HKLM\reg6 reg6.reg
echo.
```

On the above piece of code used in the Windows batch file, the statement ‘echo’ is used just for printing and outputting purposes. It does not perform any action on the system itself.

```
reg.exe LOAD HKLM\reg2 "C:\Documents and Settings\Bruno\NTUSER.DAT"
```

The above line of code is responsible for loading all the user data, which in this example is my personal user data into the reg2 key, which will be placed in the “HKey_Local_Machine” parent node.

```
reg.exe EXPORT HKLM\reg2 reg2.reg
```

On this other command we just export all of the registry key data that was just loaded into the reg2 to a file called reg2.reg. We perform steps like the one just described for various data such as: User, Security Account Manager, Software, System, and Configuration data files. We then use the below command to bundle all of the registry key data files that were created into a single file that we call registry.reg file.

```
copy reg2.reg + reg4.reg + reg6.reg + reg7.reg + reg8.reg registry.reg
```

Another feature from these scripts is to get a directory listing of the entire system drive. This will get a listing of all the files and directories that are present in the HD drive. The reason why this is important is because the system will be able to compare and find out if any files or directories were created when running the suspicious binary code. This is the Windows command that performs this action:

```
pushd C:\
dir /s /b > "%CWD%\FileSys.txt"
popd
```

The first line is responsible for saving the C:\ drive directory structure including files present into memory [29]. The *dir* command presented on the 2nd line, tells the system to display the list of files and subdirectories in a directory. The additional parameters */s /b*, specifies to the system not to include heading or summary information and to list in an alphabetical order [30]. Now this additional part of the 2nd command, > *"%CWD%\FileSys.txt",* tells the Windows system instead of showing the output on the command screen, to write the contents to file called *FileSys.txt* which is to be written to the current directory of where the batch file is running. Lastly the 3rd command, *popd,* is the one that tells the system to take all of these directories listing that are present in memory and to pop them, or in another words output them.

4.2 Directory Naming Convention

For every suspicious code that is run in the MaCod system for analysis, the system will generate reports and datasets for before and after running of the suspicious code. A naming convention for these directories of where the datasets were saved was adopted for easy identification and categorization. The datasets and report files are saved in a subdirectory in the form of "mm_dd_yyyy_xxx" to the local path: '*F:\Thesis\*'. Here is part of the code in C# that is responsible for creating a Windows process that will call the snapshot.bat scripting file, wait for it to finish running and then after that it will create the directory using the naming convention mentioned above and then it will save the files to that directory:

```
private static void CallBatchFile()
{
    Process batchProcess = new Process();
    batchProcess.StartInfo.FileName = DriveLetter.Name + BATCH_FILE_PATH;
    try
    {
        batchProcess.Start();
        batchProcess.WaitForExit();
    }
    catch (Exception e)
    {
        MessageBox.Show("Exception: " + e.Message);
    }
}
```

The above method is responsible for calling the after batch file, waiting for it to run and generating all of registry key and file system data. This piece code shown below is responsible for creating the subdirectory where the files will be saved, but before it will check if a subdirectory is already present: if it is, it will increment the number, as shown in the piece of code below:

```
private void BuildSubDirectoryPath()
{
    //start with count
    int count = 1;
    // get subdirectory to copy files
    string subDirectoryName = GetSubdirectoryPath(count);

    //while subdirectory exist
    while (Directory.Exists(subDirectoryName))
    {
```

```
                //increment counter
                count++;
                //try again and see if exists
                subDirectoryName = GetSubdirectoryPath(count);
            }

            //when it does not exist, create the subdirectory
            Directory.CreateDirectory(subDirectoryName);
            //Copy files to the specified subdirectory
            CopyFiles(subDirectoryName);
        }
```

After creating the subdirectory the program will then copy the files to that directory. Below is the piece that handles this functionality:

```
       //Copy files to the subdirectory specified
        private static void CopyFiles(string subDirectoryName)
        {
            string path =
                _snapshotStatus == SNAPSHOT_STATUS.BEFORE
                ? BEFORE_DATA_FILES_DIRECTORY
                : AFTER_DATA_FILES_DIRECTORY;

            //Get list of all files that need to be copied
            string[] files = Directory.GetFiles(DriveLetter.Name + path);

            foreach (string f in files)
            {
                try
                {
                    //copy the file to the subdirectory specified
                    File.Copy(f, subDirectoryName + @"\" + Path.GetFileName(f));
                }
                catch (IOException e)
                {
                    Console.WriteLine("IoException: " + e.Message);
                }
            }
        }
```

The code seen above is responsible for building the subdirectory for all the analysis data in the form of "mm_dd_yyyy_xxx", being month, day, year, number of occurrences for that given day, respectfully that is generated by the After Batch script file. As you can see it takes many things into account, if the directory is available, if it has less than 10 subdirectories that meet the date of when the code is run. We do this to facilitate our process of data analysis.

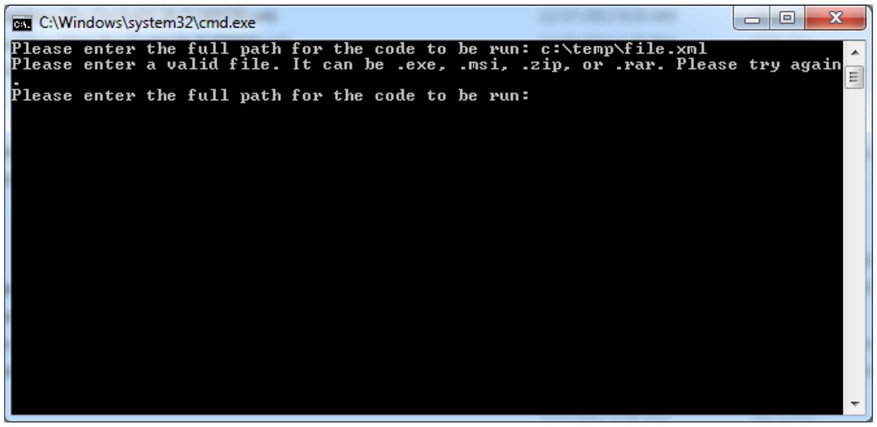
4.3 Binary Code Runner

This module is responsible for running the suspicious binary code in the Experimental Partition. This module was created using the DOS batch scripting language. As the system loads from the Experimental Partition, the binary code runner will open a Windows console window, prompting the user to provide the full path of where the suspicious code is located. Scripting programming is still very popular and an easy way to code when small functionality is need and no GUI is necessary to interact with the user. In order for this code to start every time that the system starts, this code was added to the Windows startup directory. In order to use this Windows functionality, the user just needs to copy the code that needs to be run during startup of the system to the Windows startup Directory.

To enter the path of the binary code that he wishes to run, the user is allowed to choose one of the four types of binary code of the file types shown below:

- .exe
- .msi
- .rar
- .zip

The .exe and .msi (Microsoft Installer) types of files are similar in that they both install some code onto a system. The .rar and .zip files are compressed files. When the .rar or .zip files are run, the script will uncompress the file to the 'C:\Code\' directory and will search for the .exe or .msi file located in this directory or its subdirectories. If for any reason the user tries to run a different file, the user will get a message asking him to enter a valid file, as is shown below in Figure 4.3.1:



```
Please enter the full path for the code to be run: c:\temp\file.xml
Please enter a valid file. It can be .exe, .msi, .zip, or .rar. Please try again
.
Please enter the full path for the code to be run:
```

Figure 4.3.1 – Unacceptable File Type Message

This will repeat until the user enters one of the allowed file types. After an acceptable file type is entered the system will run the code and it will restart the system after a period of 300 seconds, or five minutes, as it is shown by the code below:

```
:DONE
::RESTART THE COMPUTER
shutdown.exe -r -t 300
exit
```

4.4 RegKeg

As previously stated experimental data was collected through the use of a Windows batch script file. This data included all system and registry files before and after the possibly infected or malicious code was run. The purpose of RegKeg was to compare these batch files and run multiple experiments to determine if the code run on the system was in fact malicious.

RegKeg, written in Windows C# utilizing Visual Studio 2010 as the integrated development environment (IDE), is a menu driven Graphical User Interface (GUI) application, utilizing multiple threads to separate the user interface updates from the file processing and analysis to enable the GUI content to be update while the processing of the tasks are occurring. The RegKeg program consists of the following modules:

- Create the Before and After Key Dataset
- Build Suspicious Dataset File
- Build General Statistics Data Points
- Before and After Dataset Analysis and Reporting
- Classification of Unknown Binary Code

4.4.1 Create the Before and After Key Dataset

This is the module that calls the Windows batch script file, snapshot.bat. This module is used to create both the before and the after dataset since they are similar to each other. In order to share the same module, when the user clicks on the menu to create the before key dataset, we pass in a snapshot status variable that will read “before”. By doing this we are able to let the program know the correct batch file that it needs to call, while still re-using the same code to generate the after dataset as well. Below is code that will allow this functionality:

```
 string path =
_snapshotStatus == SNAPSHOT_STATUS.BEFORE
? BEFORE_BATCH_FILE_PATH
: AFTER_BATCH_FILE_PATH;
```

This module also creates the subdirectories using the naming convention described in the previous section, and copies the before and after dataset files that are generated to the subdirectory that was just created by the module. After this process is complete, an alert message will be displayed to the user.

4.4.2 Build Suspicious Dataset File

In this module RegKeg will grab all of the keys that are in the “nonPresent.txt” report files. This report is generated by the program for every binary code that is run in the system by comparing the before and after datasets. For every key that is found in the after dataset and not found in the before dataset, an entry will be logged to the “nonPresent.txt” report file. To run this module the user needs to enter the path for the parent directory of where the datasets are found. For any reason, if the user tries to run the program without providing this parameter, the program will give an alert message to the user reminding him to enter the appropriate information needed to run this module, as show in figure 4.4.1 below:

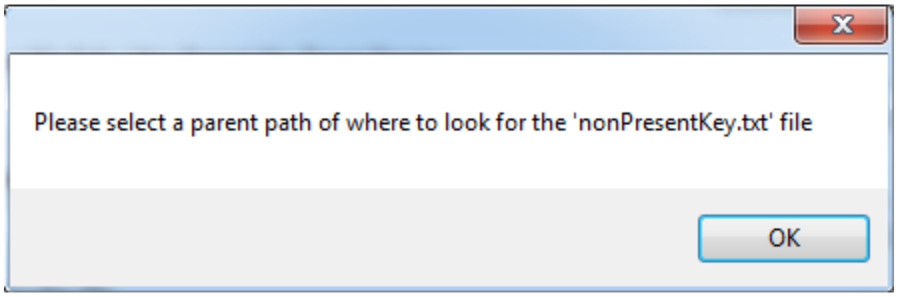

Please select a parent path of where to look for the 'nonPresentKey.txt' file

Figure 4.4.1 – Parent Path Alert Message

After the correct path is provided the program will search for all of the directories that contains the desired "nonPresentKey.txt" report file, by using the below code:

```
string[] fileNames = Directory.GetFiles(this._parentDirectoryPath, "nonPresentKeys.txt",
SearchOption.AllDirectories);
```

The above code will return an array of file objects, containing the full path of where the files are located, its name and other useful properties related to the files. The program then will loop through the list of files and will open the file to be read using the below code:

```
string[] keys = System.IO.File.ReadAllLines(filename);
```

At this point the program will loop through all of the registry keys in the dictionary, if any of these keys are not present in the dictionary of unique keys it will then be added to this dictionary. If the current key being read is present in the dictionary of unique keys it will then be skipped and it will be added to the dictionary of duplicate keys. After completing this process, the module will generate a report file called "master_suspicious_keys.txt" that will contain the list of registry keys that were found in the after datasets and that were not present in the before datasets.

4.4.3 Build General Statistics Data Points

This module is responsible for aggregating all of the statistics data points into a single easy to read comma separated value (CSV) file that can be opened with Microsoft Excel software. To initiate this module the user needs to click on the menu file button that reads "Build General Statistics Data Points" and also provide the path to the parent directory of where all of the "statistics.txt" report files are found. For every dataset that was compared and analyzed in the system, a statistics report file was generated and saved for later analysis and reporting. If the user fails to provide the path for this directory, the program will display a message to the user that reads: "Please select a parent path of where to look for the 'statistics.txt' file". The program will not proceed until this parameter is provided.

After providing the parameters necessary to run this module, the program will proceed by going through the parent directory path that was provided and will search for all the files within that directory and its subdirectories that have the name of "statistics.txt". After locating all of these files, the content of these files will be read into memory and later on used to generate a single file containing each value separated by a comma. Below is the code that performs these actions:

```
List<string> dataFiles = GetAllDataFiles();
GetDataFromFiles(dataFiles);
WriteGrandTotal(GRAND_TOTAL_STATISTICS);
WriteCsvStatisticsData(GetStatisticsData(), this._parentDirectoryPath + @"statistics.csv");
```

After completing its tasks, the program will alert the user that the code has finished by displaying an alert message.

4.4.4 Before and After Dataset Analysis and Reporting

In this module RegKeg will compare the before and after datasets. In order to order to run this module the user has to provide five parameters:

- Path for Before Data Files
- Path for After Data Files
- Path for Outputting Report Files
- Path for Master Suspicious Key File
- Path for Scoring Key File

If the user fails to provide any of these five parameters before attempting to run the comparison module, the program will not proceed and it will display alert messages to the user, as it is shown in the code below:

```
            //path for before dataset
            if (txtBeforeFilePath.Text == "") {
                MessageBox.Show("Please choose directory path for the before data
points", "Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }

            //path for after dataset
            if (txtAfterFilePath.Text == "") {
                MessageBox.Show("Please choose directory path for the after data points",
"Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }
            if (txtOutputFilePath.Text == "") {
                MessageBox.Show("Please choose directory path for outputting the report 
files", "Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }

            //path for master suspicious key file
            if (txtMasterSuspiciousKeyFiles.Text == "") {
                MessageBox.Show("Please choose directory path for the Master Suspicious 
Key file", "Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }

            //path for scoring file
            if (txtScoringFile.Text == "") {
                MessageBox.Show("Please choose directory path for the Scoring Key file", 
"Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }
```

The above code will check to see if the input parameters were valid, if they are not valid, the program will return without performing the rest of the tasks for this comparison module. After providing the appropriate parameters necessary to run this module, the program will then search in the before directory path that was provided by the user, and it will search for the registry and the file system files. The way that this is done is by searching the directory and filtering the search by giving the file extension as its criteria. For the Registry files, we use .reg and for the file system, we use the .txt file extension. This is the code that performs this task:

```
string[] regFiles = Directory.GetFiles(txtBeforeFilePath.Text,"*.reg");
string[] fileSystem = Directory.GetFiles(txtBeforeFilePath.Text, "*.txt");
```

After locating the registry file, the program will read the registry file into memory and then parse its data. The way that a registry key is presented is in the form as shown below:

```
[HKEY_LOCAL_MACHINE\reg2\Control Panel]
```

"Opened"=dword:00000001

The registry key is shown in between the square brackets, [ ], and the next line is the value for this registry key. What the above registry key is telling the Windows system is that the Control Panel window was opened. This is true because the "Opened" property has the value of 1. So the way that the program identifies a registry key is by checking the line that is being read if it starts and ends with a square bracket as it is shown in the piece of code below:

```
if (line.StartsWith("[") && line.EndsWith("]"))
{
        //more code here
}
```

Until the program finds another registry key, the line(s) that will be read are considered the value for the previous key that was processed earlier. The following code segment performs this functionality:

```
if (line == "" || lineCounter == 0)
{
       lineCounter++;
       continue;
}

if (line.StartsWith("[") && line.EndsWith("]"))
{
       try
       {
              if(key != null)
              {
                    this._registryDictionary.Add(key, value);
                    value = "";
              }
       }
       catch (Exception e)
       {
              Console.WriteLine("Exception: " + e.Message);
       }
       key = line;
 }
 else
 {
       value += line + "\n";
}
```

After processing the registry keys, the next step will be to read the system file. The system file was generated by the snapshot.bat batch script file and will contain all the files and directories for the system. RegKeg will then read this file and create a list of files. After finishing creating the before dictionary and the list of files, the program will perform the same task but at this time, targeting the after dataset. It will create the after registry key dictionary and the list of system files for the after dataset.

The next step of the process is to read the scoring data file into memory and to create a dictionary containing the registry key as the key and the frequency score that represents the number of times that this key was found in the training datasets. The file is in the form of: Key | Frequency. The pipe delimiter separates the key from the frequency. This is the segment of code that performs this action:

```
values = line.Split('|');
scoringSystem.Add(values[0], new FrequencyScore(
                                                Convert.ToInt32(values[1]),
                                                Convert.ToInt32(values[2])
                                           ));
```

The next task in the list is to compare the before and after datasets. This is done by going through each and every registry key in the before and after dictionaries. The first thing it does is to check to see if it the key is available in the before Dictionary. If it is available it will look to see if the values match or not. If they match we do not need to do anything otherwise the key and the different values will be written to the “Analysis.txt” report file. Now if the key is not contained in the lookup dictionary, the program will check to see if this key is available in the frequency score dictionary set, if this is the case, it will add this score to the dictionary of scoring results, to later be used to generate the statistics data points. If the key is not contained in the frequency score dictionary, then the program will log this information to the "nonPresentKeys.txt" report data file. Below is the code that performs the described actions:

```
if (lookup.ContainsKey(key))
{
     //Values are different
     if (lookup[key] != target[key])
     {
         if (targetDict == Target.After)
         {
             //Write to Analysis.txt report file
             writer.WriteLine(key + System.Environment.NewLine);
             writer.WriteLine("BEFORE: " + System.Environment.NewLine + lookup[key]);
             writer.WriteLine("AFTER: " + System.Environment.NewLine + target[key]);
             writer.WriteLine("-------------------------------------------");

             this._mismatchCount++;
          }
      }
}
else
{
     //the key is not available in the lookup Dictionary

    //this key is found in the list of clean keys
    if (_cleanKeys.ContainsKey(key))
        continue;

    if (_frequencyScore.ContainsKey(key))
    {
        //it is found in the frequency score dictionary
        FrequencyScore scoring = this._frequencyScore[key];
        this._score += scoring.Score;
        scoringResults.Add(scoring.Score);
        result.WriteLine(key + "=" + scoring.Score);

        if (scoring.Score > maxScore)
             maxScore = scoring.Score;
    }
    else
    {
           //not found in the frequency dictionary
          //log it to the nonPresentKeys.txt report data file
          notPresentWriter.WriteLine("Key: " + key);
          notPresentWriter.WriteLine("Value: " + target[key]);
          notPresentWriter.WriteLine("----------------------------------------");

          //write to list of non present keys - build dictionary of suspicious keys
          nonPresentKeys.WriteLine(key);
          this._notPresentOnTargetDictionary++;
     }
}
```

After going through all the keys, the last task for this module is to then calculate all the statistical data points and it will log them into the “statistics.txt” report file. The statistical points used are:

- Average
- Variance
- Standard Deviation

Before the module tries to do the calculations it will check to see if the score integer is greater than zero. If this is the case, it will proceed with all the calculation points, as shown below:

```
if (this._score > 0)
{
     result.WriteLine("TotalScore=" + this._score);
     statistics.WriteLine("TotalScore=" + this._score);

     double average = GetAverage(this._score, this._frequencyScore);
     result.WriteLine("Average=" + average);
     statistics.WriteLine("Average=" + average);

     double variance = GetVariance(scoringResults);
     result.WriteLine("Variance=" + variance);
     statistics.WriteLine("Variance=" + variance);

     double standardDeviation = GetStandardDeviation(variance);
     result.WriteLine("StandardDeviation=" + standardDeviation);
     statistics.WriteLine("StandardDeviation=" + standardDeviation);
}
```

4.4.5 Classification of Unknown Binary Code

This is the final module that the user will run for each binary code that he wants to classify. In order to run this module the user needs to provide only one parameter, which is the path for the "statistics.txt" data file for the binary code that he wishes to classify. After providing this parameter, the program will proceed by reading the statistics.txt data file and storing the information in a custom built object.

```
StatisticsReader reader = new StatisticsReader(txtOutputFilePath.Text);
```

After reading the contents of the statistics report, the module will then pass in this information to another object, which will then analyze the data and attempt to classify the binary code as being malicious or benign. The classification of the code is based on two data points that we learn during the experiments and the training datasets analysis. These points are the mean and the standard deviation of the current dataset in relation to the entire dataset that was gathered during the training of the program. Here is the code that performs the classification:

```
private CODE_CLASSIFICATION DetermineCodeClassification()
{
    if (this._statisticsData.Average > 0.02)
    {
        if (this._statisticsData.StandardDeviation > 3.6)
        {
            this._classificationPercentage = 100.0f;
            return CODE_CLASSIFICATION.MALICIOUS;
        }
        else
        {
            this._classificationPercentage = 75;
            return CODE_CLASSIFICATION.MALICIOUS;
        }
    }

    if (this._statisticsData.Average <  0.02)
    {
        if(this._statisticsData.StandardDeviation < 3.6)
        {
            this._classificationPercentage = 100.0f;
            return CODE_CLASSIFICATION.CLEAN;
        }
        else
        {
            this._classificationPercentage = 75;
            return CODE_CLASSIFICATION.CLEAN;
        }
    }

    this._classificationPercentage = 0;
    return CODE_CLASSIFICATION.UNKNOWN;
}
```

In this chapter we present the system implementation for the MaCod System. With this system and all the steps presented above, the user is able to run a suspicious binary code and within minutes be able to classify the binary code as being benign or malicious.

CHAPTER 5: RESULTS

This chapter focuses on the results of running a suspicious binary code in the MaCod System. It describes the samples used to train the RegKeg system and the experimental sets that were used to determine the results of the experiment. It will conclude with the results of this approach.

5.1 Training RegKeg

This section describes the process that was used to train RegKeg. In order to train the program, 66 sample datasets were collected and run against the system. Out of those 66 samples, 32 of them were known malicious code samples that were collected during the period of February of 2012 until March of 2013 from the website: http://www.MalwareBlacklist.com. The MalwareBlacklist.com website indexes URLs where malicious code samples are found. This website is a public list of links to known malicious code executables. It is one of the most frequently updated malicious code indexes available. For this project, the intention was to classify fairly recent binary code of the following types:

- .exe
- .msi,

- .rar,
- .zip.

The other 34 samples used to train the system were known to be non-malicious binary codes. The criteria for these benign samples were to be regular and popular used binary code popularly used by the mass population. Some of the binaries collected and run against the MaCod system were: Skype, Chrome, Winamp, ITunes, and Adobe PDF Reader. These samples were collected from the period of April of 2013 until September of 2013. For the 66 samples that were collected during a period of about a year and a half, the datasets were manually analyzed for each and every log and report file that were generated by the RegKeg program. Table 5.1.1 shows the breakdown of samples and their classifications for the training process of the RegKeg program:

Sample	Number of Samples
Malicious samples	34
Benign samples	32

Table 5.1.1 – Training Sample Breakdown

5.1.1 Compare Before and After Datasets in Training Mode

The first step to training the system is to run the before and after datasets for each and every training sample that is available. In this process the user selects the training mode option in RegKeg program as shown in Figure 5.1.1 below:

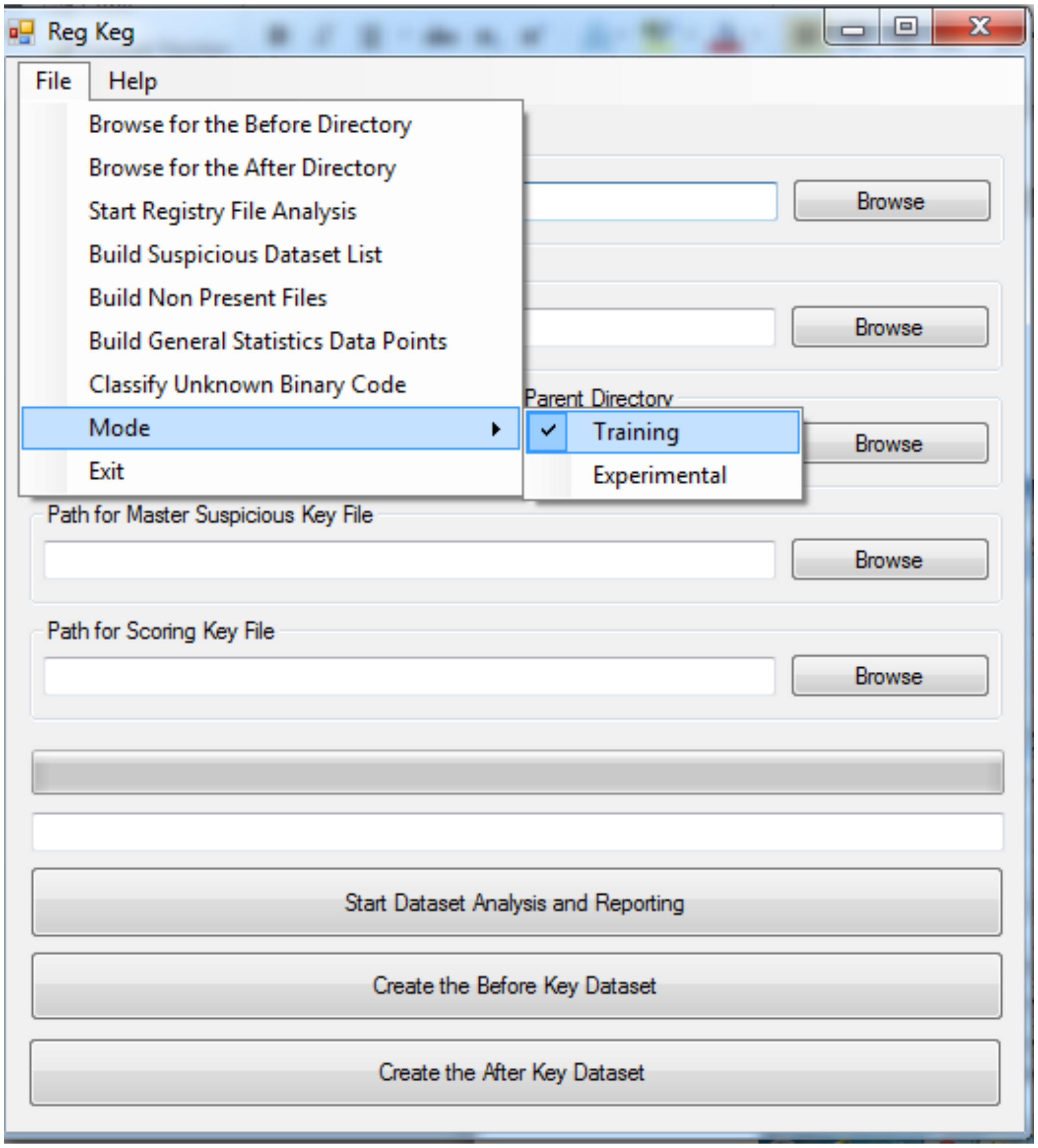

Reg Keg
File
Help
Browse for the Before Directory
Browse for the After Directory
Start Registry File Analysis
Build Suspicious Dataset List
Build Non Present Files
Build General Statistics Data Points
Classify Unknown Binary Code
Mode
Exit
Parent Directory
✓ Training
Experimental
Browse
Browse
Browse
Path for Master Suspicious Key File
Browse
Path for Scoring Key File
Browse
Start Dataset Analysis and Reporting
Create the Before Key Dataset
Create the After Key Dataset

Figure 5.1.1 – Training Mode Option

After selecting the training mode in the RegKeg program the user can “Start the Dataset Analysis and Reporting” module against the before and after datasets for every training dataset.

During this analysis and reporting module, different things occur and one of them is to log all of the registry keys that are found in the after dataset that are not present in the before dataset to the file that it is named: “nonPresent.txt”.

5.1.2 Build Suspicious Dataset List

The next step to train the system is to build the suspicious dataset list. In order to do that, the user will provide the path for the parent directory where all of the training datasets are on the disk. For every one of these datasets, a “nonPresent.txt” file will be included inside its directory. The module will then loop through each and every subdirectory that is found in the path provided by the user. The program will look for the “nonPresent.txt” file, once this file is located and read into memory, all of its data, in this case every registry key, will be written to a different file that it is called “master_suspicious_keys.txt”, keeping a tally of all of the Windows Registry keys found in all of the training sets.

After building the “master_suspicious_keys.txt” file, there is another piece of code in the same module that will read each registry key in the file and will store it in the memory of the computer using a Dictionary data structure. Before inserting the newly read registry key into the Dictionary data structure, it checks to see if the key same key has already been found and it is present in the dictionary, if this is the case, the code will retrieve this dictionary key and will increment the value for the key by one. This is done for every key that is found in the “master_suspicious_keys.txt” report. This will give RegKeg the weight of each key in respect to the training dataset. This information is kept in a separate file that it is called “scoring_keys.txt” report file.

5.1.3 Malicious Code Patterns

Based on our preliminary tests, it can be noticed that malicious codes do use registry keys to attach themselves to the operating system calls. By their nature, malicious codes use as few resources as possibles. It can be noticed that the number of registry keys used by each malicious code is rather limited.

On the other hand, the registry keys used by the malicious codes are those keys most frequently used by all other applications. This fact will help malicious codes to disguise themselves as regular code, and at the same time utilize most popular operating system calls.

In our project design, we need to provide an appropriate measure that will be based on both registry keys frequencies and the number of specific registry keys used by the application to be tested. We are expecting that the malicious application would in general use just a few of the most frequent registry keys.

5.1.4 Compare Before and After Datasets in Experimental Mode

The third step is similar to the first step but now the user will switch the mode to the “Experimental” option. In addition to this the user needs to provide two additional parameters, the path for the “master_suspicious_keys.txt” file and the path for the “scoring_keys.txt” file. The first thing the program will do is to read the “master_suspicious_keys.txt” line by line and insert the registry keys into a Dictionary data structure. Before it inserts the registry key into the dictionary, it will check to see if it is already present, and if this is the case, it will then increment the frequency for the given key by one.

After creating the frequency dictionary, the module will compare the before and after registry key files and for every key that it is found in the after dataset and that it is not present in the before dataset, it will then retrieve the frequency score for this registry key from the frequency Dictionary and it will add value to a running total for all of the keys. This is the formula that was described above in words:

```
//Retrieve frequency from the Dictionary for the given key
FrequencyScore scoring = this._frequencyScore[key];
//Add this value to a running total of frequency score
this._score += scoring.Score;
```

One can say that scoring is based on the frequencies

scoring = key_frequency

total_score = ∑ Score

After this process is completed and all of the keys from the after dataset are compared to the before dataset, the module will then calculate the weight by key value. This is done by dividing the total running score by the number of keys present in the frequency dictionary. Below is the formula used to calculate this weight by key value:

```
//divide the running total by the number of keys in the frequency dictionary.
double weightByKey = totalFrequencyScore / (double)frequencyScoreDictionary.Count;
```

Another statistical measure that is calculated is the standard deviation of the weight by key for each dataset. All of this information is then saved to a report file called “statistics.txt”.

5.1.5 Build Statistical Data Points File

In this module the user will provide one parameter, which is the path of the parent directory of where all of the training data samples are stored with their report files. Once this is provided the module will then search this directory and its subdirectories for the report files called “statistics.txt”. Upon finding a “statistics.txt” file it will read all of its data and write this data to a comma separated value (csv) file having the following parameters:

- File path/name
- A flag indicating if the file is malicious or not
- Weight by key for this file
- The max score found when running the comparison for this file
- The standard deviation of weight by key
- The total score found
- The variance of weight by key

This module will gather all the data into a single file and will facilitate the analysis by the user when trying to establish the threshold values needed to determine if a binary code is malicious code or not.

5.2 Threshold Values

After training Regkeg using the training samples and building the "statistics.csv" the user needs to carefully review this report file. By carefully reviewing the statistics.csv file, the file that gathers all the data for all of the "statistics.txt" files from all of the samples into a single place, for the purpose of this project and for the training samples used to train RegKeg, it was discovered that the malicious codes usually have a weighted total score greater than 0.02 keys per code and a standard deviation of weight by key greater than 3.6 keys per code in reference to the training data that was collected for this experiment. And on the other hand this is the opposite for the benign binary codes. They usually stay below that with a weighted total score of less than 0.02 and the standard deviation less than 3.6 of weight by key. Table 5.2.2 shows these statistical data points broken down by the binary code classification:

Binary Code Classification	Weight by Key	Standard Deviation
Malicious samples	> 0.02	> 3.6
Benign samples	<= 0.02	<= 3.6

Table 5.2.2 – Statistical Points for Binary Classification

Good binary codes usually do not perform a lot of changes to the system so they have a weighted total score less than 0.02 and the standard deviation less than 3.6 in reference to the training samples. These values are the threshold values that were learned during the manual process of analyzing each dataset sample. These values worked for the majority of the dataset samples that were gathered during the experiment.

5.3 Experimental Sample Datasets

After reviewing these 66 sample datasets and learning the threshold values that would work for this research project, we ran the experiment on 30 sample datasets to test the quality of the program, and to see if the program would be able to detect and classify the unknown binary code. Out of the 30 experimental samples that were used, 15 of them were malicious codes and the other 15 were benign code. Table 5.3.1 shows the results for the classification for the binary codes:

Total Classifications	Status of Classification	% of Classification
3	Incorrect	10
27	Correct	90

Table 5.3.1 – Classification Status Results

For the three incorrect classifications, the program classified them as being malicious codes, but in really they were benign code. The remainders were correct, all of the malicious code samples were classified as being malicious and twelve of the benign samples were classified correctly, with only three being classified incorrectly as malicious.

5.4 Performance

The other important information that we learned was that for all the experimental sample datasets, the duration of the process of comparing the before and after dataset and for the generation of the reports and statistical files was done in less than one minute. Table 5.4.1 shows the breakdown of the analysis duration period and in reference to its effectiveness:

Total Classifications	Duration (in mins) to Generate Results Data	Status of Classifications
27	Less than 1 minute	Correct
3	Less than 1 minute	Incorrect

Table 5.4.1 – Duration/Effectiveness of Status Results

The results shown above confirm the effectiveness of the MaCod system for the security community. By following the steps and using the modules the way that they were described in the previous section, we discovered that is possible to classify these kinds of binary codes using the MaCod System and have a success rate of 90%. The system is able to analyze unknown viruses that we have seen proliferate more recently. These results confirm the effectiveness of the MaCod system in real-world environments.

CHAPTER 6: FUTURE WORK

There are several enhancements that can be done to the MaCod System in the near future. One of these features would be to analyze and compare the system file that is gathered during the before and after snapshot of the system. This would give the research project another way to gather statistical data and to use as a point of classification of the binary code as being malicious or non-malicious.

Another area that could be done to improve is to automate the gathering of the experimental datasets. This could be done with a batch file that would run independently in the Experimental partition to download the desired binary code and then call the Binary Code Runner job to initiate the installation of the code and halt the system after five minutes. After that, it would restart the system in the Running Partition and would call the snaphot.bat file to gather the after dataset and to save its contents. After performing this task it would also restart the system now in the Experimental partition and therefore repeat the process. This would enable us to have a bigger experimental dataset to train the system. This is very helpful since during the manual process, it usually took about thirty to forty minutes to gather each dataset.

APPENDICES

APPENDIX A: SOURCE CODE

Before Batch Script:

```
@echo off
echo Enter /? as command line argument for syntax summary: Before /?
echo.
echo.
if "%1"=="/?" goto Syntax
if "%1"=="-?" goto Syntax
if "%1"=="?" goto Syntax

echo Capturing system drive contents...
rem Setup a "Before" directory
rem mkdir Before
rem cd Before
mkdir %cd:~0,2%\Thesis\Before-After\Before
cd %cd:~0,2%\Thesis\Before-After\Before
set CWD=%cd%

rem Go to root of system drive and get dir listing of entire drive
rem pushd %SystemDrive%\
pushd C:\
dir /s /b > "%CWD%\FileSys.txt"
popd

echo Capturing Registry contents...
rem Now grab a copy of the registry
rem cmd /C regedit /e .\registry.reg
echo.


echo Loading Reg2 Hive - HKEY_CURRENT_USER KEY
reg.exe LOAD HKLM\reg2 "C:\Documents and Settings\Bruno\NTUSER.DAT"
echo Exporting Reg2 Hive - HKEY_CURRENT_USER KEY
reg.exe EXPORT HKLM\reg2 reg2.reg
echo.

echo Loading Reg4 Hive - HKEY_LOCAL_MACHINE - SAM KEY
reg.exe LOAD HKLM\reg4 "C:\WINDOWS\system32\config\sam"
echo Exporting Reg4 Hive - HKEY_LOCAL_MACHINE - SAM KEY
reg.exe EXPORT HKLM\reg4 reg4.reg
echo.
```

```
echo Loading Reg6 Hive - HKEY_LOCAL_MACHINE - SOFTWARE KEY
reg.exe LOAD HKLM\reg6 "C:\WINDOWS\system32\config\software"
echo Exporting Reg6 Hive - HKEY_LOCAL_MACHINE - SOFTWARE KEY
reg.exe EXPORT HKLM\reg6 reg6.reg
echo.

echo Loading Reg7 Hive - HKEY_LOCAL_MACHINE - SYSTEM KEY
reg.exe LOAD HKLM\reg7 "C:\WINDOWS\system32\config\system"
echo Exporting Reg7 Hive - HKEY_LOCAL_MACHINE - SYSTEM KEY
reg.exe EXPORT HKLM\reg7 reg7.reg
echo.

echo Loading Reg8 Hive - HKEY_USERS KEY
reg.exe LOAD HKLM\reg8 "C:\WINDOWS\system32\config\default"
echo Exporting Reg8 Hive - HKEY_USERS KEY
reg.exe EXPORT HKLM\reg8 reg8.reg
echo.

echo Copying reg [1-9] to registry.reg file
copy         reg2.reg + reg4.reg + reg6.reg + reg7.reg + reg8.reg registry.reg
echo.

echo Unloading Reg2 Hive
reg.exe unload HKLM\reg2
echo.

echo Unloading Reg4 Hive
reg.exe unload HKLM\reg4
echo.

echo Unloading Reg6 Hive
reg.exe unload HKLM\reg6
echo.

echo Unloading Reg7 Hive
reg.exe unload HKLM\reg7
echo.

echo Unloading Reg8 Hive
reg.exe unload HKLM\reg8
echo.


echo Deleting reg [1-8] files
del reg2.reg reg4.reg reg6.reg reg7.reg reg8.reg

echo.



echo.

rem Use type to convert Unicode reg file into Ansi char set.  Depends on
rem default code page on system.  This may not work on a Unicode
rem language system like a Japanese OS version.
type registry.reg > Ansi_Registry.reg

rem For now, keep around the original Unicode version just in case ;)
REM ren registry.reg Uni_registry.reg

cd ..
echo.
```

```
echo System state captured...

exit /b

:Syntax
echo.
echo.
echo Before.bat: Creates a directory named "Before" in the current
echo             directory and creates the following three files in
echo             that directory:
echo             1) Filesys.txt   This file contains a record of all
echo                files on the system drive
echo             2) Ansi_Registry.reg   This is an ANSI format file
echo                containing the current registry contents.
echo             3) Uni_Registry.reg    This is a Unicode format file
echo                containing the current registry contents.
echo.
echo Usage:      Before
echo.
echo NOTE: Before.bat has a companion batch file called After.bat that
echo performs the same operations and then calls Windiff to facilitate
echo analysis of the system's "before" and "after" states.
echo.
echo.
echo.
echo.
echo.
```

After Batch Script:

```
@echo off
echo Enter /? as command line argument for syntax summary: After /?
echo.
echo.
if "%1"=="/?" goto Syntax
if "%1"=="-?" goto Syntax
if "%1"=="?" goto Syntax

echo Capturing system drive contents...
rem Setup a "After" directory
mkdir %cd:~0,2%\Thesis\Before-After\After
cd %cd:~0,2%\Thesis\Before-After\After
set CWD=%cd%

rem Go to root of system drive and get dir listing of entire drive
rem change %SystemDrive% to Hard Drive Letter
rem pushd %SystemDrive%\
pushd C:\
dir /s /b > "%CWD%\FileSys.txt"
popd

echo Capturing Registry contents...
echo.
rem Now grab a copy of the registry
rem -- cmd /C regedit /e .\registry.reg

echo Loading Reg2 Hive - HKEY_CURRENT_USER KEY
reg.exe LOAD HKLM\reg2 "C:\Documents and Settings\Bruno\NTUSER.DAT"
echo Exporting Reg2 Hive - HKEY_CURRENT_USER KEY
reg.exe EXPORT HKLM\reg2 reg2.reg
echo.

echo Loading Reg4 Hive - HKEY_LOCAL_MACHINE - SAM KEY
reg.exe LOAD HKLM\reg4 "C:\WINDOWS\system32\config\sam"
echo Exporting Reg4 Hive - HKEY_LOCAL_MACHINE - SAM KEY
reg.exe EXPORT HKLM\reg4 reg4.reg
echo.

echo Loading Reg6 Hive - HKEY_LOCAL_MACHINE - SOFTWARE KEY
reg.exe LOAD HKLM\reg6 "C:\WINDOWS\system32\config\software"
echo Exporting Reg6 Hive - HKEY_LOCAL_MACHINE - SOFTWARE KEY
reg.exe EXPORT HKLM\reg6 reg6.reg
echo.

echo Loading Reg7 Hive - HKEY_LOCAL_MACHINE - SYSTEM KEY
reg.exe LOAD HKLM\reg7 "C:\WINDOWS\system32\config\system"
echo Exporting Reg7 Hive - HKEY_LOCAL_MACHINE - SYSTEM KEY
reg.exe EXPORT HKLM\reg7 reg7.reg
echo.

echo Loading Reg8 Hive - HKEY_USERS KEY
reg.exe LOAD HKLM\reg8 "C:\WINDOWS\system32\config\default"
```

```
echo Exporting Reg8 Hive - HKEY_USERS KEY
reg.exe EXPORT HKLM\reg8 reg8.reg
echo.

echo Copying reg [1-9] to registry.reg file
copy       reg2.reg + reg4.reg + reg6.reg + reg7.reg + reg8.reg registry.reg
echo.

echo Unloading Reg2 Hive
reg.exe unload HKLM\reg2
echo.

echo Unloading Reg4 Hive
reg.exe unload HKLM\reg4
echo.

echo Unloading Reg6 Hive
reg.exe unload HKLM\reg6
echo.

echo Unloading Reg7 Hive
reg.exe unload HKLM\reg7
echo.

echo Unloading Reg8 Hive
reg.exe unload HKLM\reg8
echo.

echo Deleting reg [1-8] files
del reg2.reg reg4.reg reg6.reg reg7.reg reg8.reg

echo.

rem Use type to convert Unicode reg file into Ansi char set.  Depends on
rem default code page on system.  This may not work on a Unicode
rem language system like a Japanese OS version.
type registry.reg > Ansi_Registry.reg

rem For now, keep around the original Unicode version just in case ;)
rem ren registry.reg Uni_registry.reg

cd ..
echo.
echo System state captured...

exit /b

:Syntax
echo.
echo.
echo After.bat: Creates a directory named "After" in the current
echo            directory and creates the following three files in
echo            that directory:
echo            1) Filesys.txt   This file contains a record of all
echo               files on the system drive
echo             2) Ansi_Registry.reg   This is an ANSI format file
echo                containing the current registry contents.
echo             3) Uni_Registry.reg    This is a Unicode format file
echo                containing the current registry contents.
echo             When the system state capture has completed, Windiff
echo             is automatically started to facilitate the analysis of
echo             of FileSys.txt and Ansi_Registry.reg files to look for
echo             possible changes to the system state which will cause
echo             certification failure.  Consult Windiff help for directions
echo             on how to use Windiff.
echo             Windiff note: double click on the file name and use F7/F8
echo                           to quickly move to file differences.
echo                           The Outline button returns you to the file
echo                           overview window.
echo.
echo Usage:      After
echo.
echo NOTE: After.bat has a companion batch file called Before.bat that
echo should be run prior to any Test Cases being performed as per the
echo U3 smart Application Certification Self Test document.
echo.
echo.
echo.
echo.
echo.
```

Binary Code Runner (Batch Script):

```
@ECHO OFF

SET rar=c:\winrar\RAR.exe
SET unrar=c:\winrar\UnRAR.exe
SET extracted_path=c:\code\

del %extracted_path%\*.* /s /f /q

SET "INPUT_FILE_PATH="
:INPUT
SET /P INPUT_FILE_PATH=Please enter the full path for the code to be run:
if /I "%INPUT_FILE_PATH:~-3%" == "rar" (
      %unrar% x %INPUT_FILE_PATH% %extracted_path%
      FOR /R %extracted_path% %%G IN (*.exe) DO START %%G
      GOTO DONE
)

if /I "%INPUT_FILE_PATH:~-3%" == "zip" (
      UNZIP %INPUT_FILE_PATH% -d %extracted_path%
      FOR /R %extracted_path% %%G IN (*.exe) DO START %%G
      GOTO DONE
)

if /I "%INPUT_FILE_PATH:~-3%" == "exe" (
      START %INPUT_FILE_PATH%
      GOTO DONE
)

if /I "%INPUT_FILE_PATH:~-3%" == "msi" (
      START %INPUT_FILE_PATH%
      GOTO DONE
)

ECHO Please enter a valid file. It can be .exe, .msi, .zip, or .rar. Please
try again.
goto input

:DONE
::RESTART THE COMPUTER
shutdown.exe -r -t 300
exit
```

RegKeg (.NET/C# Program):

```
/*
 * File:    AssemblyInfo.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System.Reflection;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("RegKeg")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyProduct("RegKeg")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2011")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("ae1edc04-412b-4cdc-b3a4-a9d24e4fc8fe")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

```
<?xml version="1.0" encoding="utf-8"?>

<!--
/*
* File:    Resources.resx
* Project: Master's Thesis
* Author:  Bruno Nader
* Adviser: Dr. Hrvoje Podnar
* Date:    May 20, 2014
*/
-->
<root>
  <!--
    Microsoft ResX Schema

    Version 2.0

    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:

    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
Framework object]</value>
        <comment>This is a comment</comment>
    </data>

    There are any number of "resheader" rows that contain simple
    name/value pairs.

    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.

    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:

    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.

    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0, 
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0, 
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
</root>


/*
 * File:    Resources.Designer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.1022
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace RegKeg.Properties {
    using System;
    
    
    /// <summary>
    ///   A strongly-typed resource class, for looking up localized strings, etc.
    /// </summary>
    // This class was auto-generated by the StronglyTypedResourceBuilder
    // class via a tool like ResGen or Visual Studio.
    // To add or remove a member, edit your .ResX file then rerun ResGen
    // with the /str option, or rebuild your VS project.

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyT
ypedResourceBuilder", "4.0.0.0")]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal class Resources {
        
        private static global::System.Resources.ResourceManager resourceMan;
        private static global::System.Globalization.CultureInfo resourceCulture;


[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance"
, "CA1811:AvoidUncalledPrivateCode")]
        internal Resources() {
        }

        /// <summary>
        ///   Returns the cached ResourceManager instance used by this class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.Edi
torBrowsableState.Advanced)]
        internal static global::System.Resources.ResourceManager ResourceManager {
            get {
                if (object.ReferenceEquals(resourceMan, null)) {
                    global::System.Resources.ResourceManager temp = new
global::System.Resources.ResourceManager("RegKeg.Properties.Resources",
typeof(Resources).Assembly);
                    resourceMan = temp;
                }
                return resourceMan;
            }
        }

        /// <summary>
        ///   Overrides the current thread's CurrentUICulture property for all
        ///   resource lookups using this strongly typed resource class.
        /// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.Edi
torBrowsableState.Advanced)]
        internal static global::System.Globalization.CultureInfo Culture {
            get {
                return resourceCulture;
            }
            set {
                resourceCulture = value;
            }
        }
    }
}




/*
 * File:    Settings.Designer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

//------------------------------------------------------------------------------
```

```
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.1008
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace RegKeg.Properties {


    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.S
ettingsDesigner.SettingsSingleFileGenerator", "10.0.0.0")]
    internal sealed partial class Settings :
global::System.Configuration.ApplicationSettingsBase {

        private static Settings defaultInstance =
((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new
Settings())));

        public static Settings Default {
            get {
                return defaultInstance;
            }
        }
    }
}




/*
 * File:    AboutRegKeg.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Reflection;
using System.Windows.Forms;

namespace RegKeg
{
    partial class AboutRegKeg : Form
    {
        public AboutRegKeg()
        {
            InitializeComponent();
            this.Text = String.Format("About Dynamic Analysis Using a Live System");
            this.labelProductName.Text = "Registry Key Analyser";
            this.labelVersion.Text = "Version: 3.1.1";
            this.labelCopyright.Text = "Copyrights: Bruno Nader";
            this.labelCompanyName.Text = "Southern Connectictut State University";
            this.textBoxDescription.Text = "A program that analysis snapshots before and
after and uknown piece of code is run";
        }

        #region Assembly Attribute Accessors

        public string AssemblyTitle
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyTitleAttribute),
false);
                if (attributes.Length > 0)
                {
                    AssemblyTitleAttribute titleAttribute =
(AssemblyTitleAttribute)attributes[0];
                    if (titleAttribute.Title != "")
                    {
                        return titleAttribute.Title;
                    }
                }
                return
System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().CodeBase);
            }
        }

        public string AssemblyVersion
        {
            get
            {
                return Assembly.GetExecutingAssembly().GetName().Version.ToString();
            }
        }

        public string AssemblyDescription
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyDescriptionAttribute),
false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyDescriptionAttribute)attributes[0]).Description;
            }
        }

        public string AssemblyProduct
        {
            get
            {
                object[] attributes = 
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyProductAttribute), 
false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyProductAttribute)attributes[0]).Product;
            }
        }

        public string AssemblyCopyright
        {
            get
            {
                object[] attributes = 
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCopyrightAttribute), 
false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyCopyrightAttribute)attributes[0]).Copyright;
            }
        }

        public string AssemblyCompany
        {
            get
            {
                object[] attributes = 
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyCompanyAttribute), 
false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return ((AssemblyCompanyAttribute)attributes[0]).Company;
            }
        }
        #endregion

        private void okButton_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}



/*
 * File:    AboutRegKeg.Designer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    partial class AboutRegKeg
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(AboutRegKeg));
            this.tableLayoutPanel = new System.Windows.Forms.TableLayoutPanel();
            this.logoPictureBox = new System.Windows.Forms.PictureBox();
            this.labelProductName = new System.Windows.Forms.Label();
            this.labelVersion = new System.Windows.Forms.Label();
            this.labelCopyright = new System.Windows.Forms.Label();
            this.labelCompanyName = new System.Windows.Forms.Label();
            this.textBoxDescription = new System.Windows.Forms.TextBox();
            this.okButton = new System.Windows.Forms.Button();
            this.tableLayoutPanel.SuspendLayout();

((System.ComponentModel.ISupportInitialize)(this.logoPictureBox)).BeginInit();
            this.SuspendLayout();
            //
            // tableLayoutPanel
            //
            this.tableLayoutPanel.ColumnCount = 2;
            this.tableLayoutPanel.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 33F));
            this.tableLayoutPanel.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent, 67F));
            this.tableLayoutPanel.Controls.Add(this.logoPictureBox, 0, 0);
            this.tableLayoutPanel.Controls.Add(this.labelProductName, 1, 0);
            this.tableLayoutPanel.Controls.Add(this.labelVersion, 1, 1);
            this.tableLayoutPanel.Controls.Add(this.labelCopyright, 1, 2);
            this.tableLayoutPanel.Controls.Add(this.labelCompanyName, 1, 3);
            this.tableLayoutPanel.Controls.Add(this.textBoxDescription, 1, 4);
            this.tableLayoutPanel.Controls.Add(this.okButton, 1, 5);
            this.tableLayoutPanel.Dock = System.Windows.Forms.DockStyle.Fill;
            this.tableLayoutPanel.Location = new System.Drawing.Point(9, 9);
            this.tableLayoutPanel.Name = "tableLayoutPanel";
            this.tableLayoutPanel.RowCount = 6;
            this.tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10F));
            this.tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10F));
            this.tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10F));
            this.tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10F));
            this.tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 50F));
            this.tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent, 10F));
            this.tableLayoutPanel.Size = new System.Drawing.Size(417, 265);
            this.tableLayoutPanel.TabIndex = 0;
            //
            // logoPictureBox
            //
            this.logoPictureBox.Dock = System.Windows.Forms.DockStyle.Fill;
            this.logoPictureBox.Image =
((System.Drawing.Image)(resources.GetObject("logoPictureBox.Image")));
            this.logoPictureBox.Location = new System.Drawing.Point(3, 3);
            this.logoPictureBox.Name = "logoPictureBox";
            this.tableLayoutPanel.SetRowSpan(this.logoPictureBox, 6);
            this.logoPictureBox.Size = new System.Drawing.Size(131, 259);
            this.logoPictureBox.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.StretchImage;
            this.logoPictureBox.TabIndex = 12;
            this.logoPictureBox.TabStop = false;
            //
            // labelProductName
            //
            this.labelProductName.Dock = System.Windows.Forms.DockStyle.Fill;
            this.labelProductName.Location = new System.Drawing.Point(143, 0);
            this.labelProductName.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
            this.labelProductName.MaximumSize = new System.Drawing.Size(0, 17);
            this.labelProductName.Name = "labelProductName";
            this.labelProductName.Size = new System.Drawing.Size(271, 17);
            this.labelProductName.TabIndex = 19;
            this.labelProductName.Text = "Product Name: Reg Keg";
            this.labelProductName.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
            //
            // labelVersion
            //
            this.labelVersion.Dock = System.Windows.Forms.DockStyle.Fill;
            this.labelVersion.Location = new System.Drawing.Point(143, 26);
            this.labelVersion.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
            this.labelVersion.MaximumSize = new System.Drawing.Size(0, 17);
            this.labelVersion.Name = "labelVersion";
            this.labelVersion.Size = new System.Drawing.Size(271, 17);
            this.labelVersion.TabIndex = 0;
            this.labelVersion.Text = "Version: 1.3.1";
            this.labelVersion.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
            //
            // labelCopyright
            //
            this.labelCopyright.Dock = System.Windows.Forms.DockStyle.Fill;
            this.labelCopyright.Location = new System.Drawing.Point(143, 52);
            this.labelCopyright.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
            this.labelCopyright.MaximumSize = new System.Drawing.Size(0, 17);
            this.labelCopyright.Name = "labelCopyright";
            this.labelCopyright.Size = new System.Drawing.Size(271, 17);
            this.labelCopyright.TabIndex = 21;
            this.labelCopyright.Text = "Copyright: Bruno Nader";
            this.labelCopyright.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
            //
            // labelCompanyName
            //
            this.labelCompanyName.Dock = System.Windows.Forms.DockStyle.Fill;
            this.labelCompanyName.Location = new System.Drawing.Point(143, 78);
            this.labelCompanyName.Margin = new System.Windows.Forms.Padding(6, 0, 3, 0);
            this.labelCompanyName.MaximumSize = new System.Drawing.Size(0, 17);
            this.labelCompanyName.Name = "labelCompanyName";
            this.labelCompanyName.Size = new System.Drawing.Size(271, 17);
            this.labelCompanyName.TabIndex = 22;
            this.labelCompanyName.Text = "Southern Connecticut State University";
            this.labelCompanyName.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
            //
            // textBoxDescription
            //
            this.textBoxDescription.Dock = System.Windows.Forms.DockStyle.Fill;
            this.textBoxDescription.Location = new System.Drawing.Point(143, 107);
            this.textBoxDescription.Margin = new System.Windows.Forms.Padding(6, 3, 3,
3);
            this.textBoxDescription.Multiline = true;
            this.textBoxDescription.Name = "textBoxDescription";
            this.textBoxDescription.ReadOnly = true;
            this.textBoxDescription.ScrollBars = System.Windows.Forms.ScrollBars.Both;
            this.textBoxDescription.Size = new System.Drawing.Size(271, 126);
            this.textBoxDescription.TabIndex = 23;
            this.textBoxDescription.TabStop = false;
            this.textBoxDescription.Text = "A program that analysis the before and after
snapshot after running an uknown cod" +
                "e";
            //
            // okButton
            //
            this.okButton.Anchor =
((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Bottom |
System.Windows.Forms.AnchorStyles.Right)));
            this.okButton.DialogResult = System.Windows.Forms.DialogResult.Cancel;
            this.okButton.Location = new System.Drawing.Point(339, 239);
            this.okButton.Name = "okButton";
            this.okButton.Size = new System.Drawing.Size(75, 23);
            this.okButton.TabIndex = 24;
            this.okButton.Text = "&OK";
            this.okButton.Click += new System.EventHandler(this.okButton_Click);
            //
            // AboutRegKeg
            //
            this.AcceptButton = this.okButton;
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(435, 283);
            this.Controls.Add(this.tableLayoutPanel);
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.Name = "AboutRegKeg";
            this.Padding = new System.Windows.Forms.Padding(9);
            this.ShowIcon = false;
            this.ShowInTaskbar = false;
            this.StartPosition = System.Windows.Forms.FormStartPosition.CenterParent;
            this.Text = "AboutRegKeg";
            this.tableLayoutPanel.ResumeLayout(false);
            this.tableLayoutPanel.PerformLayout();
            ((System.ComponentModel.ISupportInitialize)(this.logoPictureBox)).EndInit();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.TableLayoutPanel tableLayoutPanel;
        private System.Windows.Forms.PictureBox logoPictureBox;
        private System.Windows.Forms.Label labelProductName;
        private System.Windows.Forms.Label labelVersion;
        private System.Windows.Forms.Label labelCopyright;
        private System.Windows.Forms.Label labelCompanyName;
        private System.Windows.Forms.TextBox textBoxDescription;
        private System.Windows.Forms.Button okButton;
    }
}



/*
 * File:    Action.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public enum Action
    {
        Analyze,
        BuildFiles
    }
}
```

```
<?xml version="1.0"?>
<!--
 File:    app.config
 Project: Master's Thesis
 Author:  Bruno Nader
 Adviser: Dr. Hrvoje Podnar
 Date:    May 20, 2014
-->
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
  <appSettings>
    <add key="CleanKeyFilePath" value="Thesis\RegKeg\RegKeg\resources\clean_keys.txt"/>
    <add key="MalicousKeyFilePath"
value="Thesis\RegKeg\RegKeg\resources\suspicious_keys.txt"/>
    <add key="BeforeReg_AfterReg_exception_file"
value="Targetexception_TargetBeforeReg_LookupAfterReg.log"/>
    <add key="BeforeReg_AfterReg_keynotpresent_file"
value="keyNotPresent_TargetBeforeReg_LookupAfterReg.log"/>
    <add key="AfterReg_BeforeReg_exception_file"
value="exception_TargetAfterReg_LookupBeforeReg.log"/>
    <add key="AfterReg_BeforeReg_keynotpresent_file"
value="keyNotPresent_TargetAfterReg_LookupBeforeReg.log"/>
    <add key="BeforeDirectoryLastPath" value="Thesis\Before-After\Before\"/>
    <add key="AfterDirectoryLastPath" value="Thesis\Before-After\After\"/>
  </appSettings>
</configuration>
```

```
/*
 * File:    CODE_CLASSIFICATION.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public enum CODE_CLASSIFICATION
    {
        CLEAN,
        MALICIOUS,
        UNKNOWN
    }
}
```

```
/*
 * File:    DataAnalyzer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public class DataAnalyzer
    {
        #region Private Fields

        private StatisticsData _statisticsData;
        private CODE_CLASSIFICATION _codeClassification;
        private float _classificationPercentage;

        #endregion

        #region Properties

        public StatisticsData StatisticsData
        {
            get { return this._statisticsData; }
            set { this._statisticsData = value; }
        }

        public CODE_CLASSIFICATION CodeClassification
        {
            get { return this._codeClassification; }
            set { this._codeClassification = value; }
        }

        public float ClassificationPercentage
        {
            get { return this._classificationPercentage; }
            set { this._classificationPercentage = value; }
        }

        #endregion

        #region Constructor(s)

        public DataAnalyzer(StatisticsData statisticsData)
        {
            this._classificationPercentage = 0;
            this._statisticsData = statisticsData;
            this._codeClassification = DetermineCodeClassification();
        }

        #endregion

        #region  Private Method(s)

        private CODE_CLASSIFICATION DetermineCodeClassification()
        {
            if (this._statisticsData.Average > 0.02)
            {
                if (this._statisticsData.StandardDeviation > 3.6)
                {
                    this._classificationPercentage = 100.0f;
                    return CODE_CLASSIFICATION.MALICIOUS;
                }
                else
                {
                    this._classificationPercentage = 75;
                    return CODE_CLASSIFICATION.MALICIOUS;
                }
            }

            if (this._statisticsData.Average <  0.02)
            {
                if(this._statisticsData.StandardDeviation < 3.6)
                {
                    this._classificationPercentage = 100.0f;
                    return CODE_CLASSIFICATION.CLEAN;
                }
                else
                {
                    this._classificationPercentage = 75;
                    return CODE_CLASSIFICATION.CLEAN;
                }
            }

            this._classificationPercentage = 0;
            return CODE_CLASSIFICATION.UNKNOWN;
        }

        #endregion
    }
}



/*
 * File:    DriveLetter.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System.IO;

namespace RegKeg
{
    public static class DriveLetter
    {
        #region Properties

        public static string Name
        {
            get;
            set;
        }

        #endregion
        #region Constructor

        static DriveLetter()
        {
            string fullPath = System.Reflection.Assembly.GetExecutingAssembly().Location;
            FileInfo f = new FileInfo(fullPath);
            Name = Path.GetPathRoot(f.FullName);
        }

        #endregion
    }
}



/*
 * File:    FileStatus.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public enum FileStatus
    {
        Before,
        After
    }
}



/*
 * File:    FileSystemComparer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;
using System.Text.RegularExpressions;

namespace RegKeg
{
    public class FileSystemComparer
    {
        public List<string> _beforeList { get; set; }
        public List<string> _afterList { get; set; }
        public string _path { get; set; }
        public List<string> _fileSystem { get; set; }
        public int _score { get; set; }
        public FileSystemComparer(List<string> beforeList, List<string> afterList, string
path)
        {
            this._beforeList = beforeList;
            this._afterList = afterList;
            this._path = Path.GetDirectoryName(path);
            GetFileSystemList();
            this._score = 0;
        }

        private void GetFileSystemList()
        {
            this._fileSystem = new List<string>();
            this._fileSystem.Add(@"C:\\Program Files\\");
            this._fileSystem.Add(@"C:\\Windows\\");
            //this._fileSystem.Add(@"C:\\Documents and Settings\\Bruno\\Application
Data\\");
        }

        public void Compare()
        {
            using (StreamWriter notPresentWriter = new StreamWriter(this._path + "\\" +
"file_system_comparer.txt",false))
            {
                List<string> notPresentFilesList = new List<string>();

                notPresentWriter.WriteLine("Files that are NOT found on the Before Data
Set System Files.");

notPresentWriter.WriteLine("=============================================================
==");
                foreach (string file in this._afterList)
                {
                    if (!this._beforeList.Contains(file))
                    {
                        notPresentFilesList.Add(file);
                        notPresentWriter.WriteLine(file);
                    }
                }

                foreach (string file in notPresentFilesList)
                {
                    try
                    {
                        if (IsWindowsDirectoryEntry(file))
                        {
                            this._score += 5;
                        }
                        else if (IsProgramFilesDirectoryEntry(file))
                        {
                            this._score += 1;
                        }
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Exception in Compare(): " + e.Message);
                    }
                }
            }
        }

        private bool IsProgramFilesDirectoryEntry(string file)
        {
            if (Regex.IsMatch(file, this._fileSystem[0]))
                return true;

            return false;
        }

        private bool IsWindowsDirectoryEntry(string file)
        {
            if (Regex.IsMatch(file, this._fileSystem[1]))
                return true;

            return false;
        }
    }
}



/*
 * File:    FileSystemReader.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;

namespace RegKeg
{
    public class FileSystemReader
    {
        #region Properties

        public string FileName { get; set; }
        public List<string> FilesList { get; set; }

        #endregion

        public FileSystemReader(string fileName)
        {
            this.FileName = fileName;
            this.ReadFile();
        }

        private void ReadFile()
        {
            this.FilesList = new List<string>();
            try
            {
                using (StreamReader streamReader = new StreamReader(this.FileName))
                {
                    string line = "";
                    while ((line = streamReader.ReadLine()) != null)
                    {
                        FilesList.Add(line);
                    }
                }
            }
            catch (Exception exception)
            {
                Console.WriteLine("Exception: " + exception.Message);
            }
        }
    }
}



/*
 * File:    Form.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;

namespace RegKeg
{
    public partial class Form1 : Form
    {
        private RegistryComparer registryComparer = null;
        private Stopwatch stopwatch;
        private Action action;

        private const string REGISTRY_FILE_FILTER = "Registry File|*.reg";
        private const string TEXT_FILE_FILTER = "Text File|*.txt";

        public delegate void poplateTextBoxDelegate(string text);
        public delegate void analysisDelegate();
        public delegate void updateProgressBar();

        public delegate void updateCompareBar();

        public Form1()
        {
            InitializeComponent();
#if DEBUG
```

```
            //LoadDefaultParameters();
#endif

        }

        private void LoadDefaultParameters()
        {
            txtBeforeFilePath.Text = @"H:\Thesis\Before-After\Before\10-17-2012_001\";
            txtAfterFilePath.Text = @"H:\Thesis\Before-After\After\";
            txtMasterSuspiciousKeyFiles.Text = @"H:\Thesis\Before-
After\Analysis\master_suspicious_keys.txt";
            txtScoringFile.Text = @"H:\Thesis\Before-After\Analysis\scoring_keys.txt";
            txtOutputFilePath.Text = @"H:\Thesis\Before-After\After\";
        }

        void populateTextBox(string text)
        {
            txtStatus.Text = text;
        }

        private void btnStartAnalysis_Click(object sender, EventArgs e)
        {
            action = Action.Analyze;
            StartClockAndAnalysis();
        }

        private void StartClockAndAnalysis()
        {
            stopwatch = Stopwatch.StartNew();

            analysisDelegate analysis_delegate = StartRegistryAnalysis;
            analysis_delegate.BeginInvoke(null, null);
        }

        private void StartRegistryAnalysis()
        {
            if (action == Action.Analyze)
            {
                if (!ValidInputParameters())
                    return;
            }

            string outputPath = GetOutputPath();

            this.Invoke(new updateProgressBar(updateProgress));

            this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[]
{"Reading Before Files..." });

            string[] regFiles = Directory.GetFiles(txtBeforeFilePath.Text,"*.reg");
            string[] fileSystem = Directory.GetFiles(txtBeforeFilePath.Text, "*.txt");

            if (regFiles.Length == 0)
            {
                MessageBox.Show("There are no Registry files in the Before directory,
please double check it and try again.");
                return;
            }

            RegistryReader beforeRegistryReader = new RegistryReader(regFiles[0]);

            this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[] {
"Creating Before Dictionary..." });
            Dictionary<string, string> beforeDic =
beforeRegistryReader.RegistryDictionary;

            FileSystemReader beforeFileSystemReader = new
FileSystemReader(fileSystem[0]);


            this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[] {
"Reading After Files..." });

            try
            {
                string[] afterRegFiles = Directory.GetFiles(txtAfterFilePath.Text,
"*.reg");
                string[] afterFileSystem = Directory.GetFiles(txtAfterFilePath.Text,
"*.txt");

                if (afterRegFiles.Length == 0)
                {
                    MessageBox.Show("There are no Registry files in the After directory,
please double check it and try again.");
                    return;
                }
                RegistryReader afterRegistryReader = new
RegistryReader(afterRegFiles[0]);

                this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[] {
"Creating After Dictionary..." });
                Dictionary<string, string> afterDic =
afterRegistryReader.RegistryDictionary;
                FileSystemReader afterFileSystemReader = new
FileSystemReader(afterFileSystem[0]);

                Dictionary<string, FrequencyScore> frequencyScoreDictionary = null;

                if (!trainingToolStripMenuItem.Checked)
                {
                    this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[]
{ "Creating Scoring Dictionary..." });
                    //scoring file path here
                    ScoringFileReader scoringFileReader = new
ScoringFileReader(txtScoringFile.Text);
                    frequencyScoreDictionary = scoringFileReader.GetScoringDictionary();
                }


                this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[] {
"Start Comparing Dictionaries..." });
                registryComparer = new RegistryComparer(beforeDic, afterDic, outputPath,
frequencyScoreDictionary);
```

```
                registryComparer.PerformComparison();

                FileSystemComparer fileSystemComparer = new
FileSystemComparer(beforeFileSystemReader.FilesList, afterFileSystemReader.FilesList,
outputPath);
                fileSystemComparer.Compare();

                this.Invoke(new updateCompareBar(updateCompareBarMethod));
                this.Invoke(new poplateTextBoxDelegate(populateTextBox), new object[] {
"Finished..." });

                WriteResultsData(outputPath);
                MessageBox.Show("Finished!!\nElapsed Comparison Time: " +
                               stopwatch.Elapsed + " seconds" +
System.Environment.NewLine +
                               "Mismatch Count: " + registryComparer.MismatchCount +
System.Environment.NewLine +
                               "Score: " + registryComparer.Score +
fileSystemComparer._score,
                               "Registry Comparison",
                               MessageBoxButtons.OK,
                               MessageBoxIcon.Information);

                OpenOutputFile(outputPath);
            }
            catch (DirectoryNotFoundException dnf)
            {
                MessageBox.Show("Directory not found. " + dnf.Message);
                return;
            }
        }

        private string GetOutputPath()
        {
            string outputPath = "";

            if (txtOutputFilePath.Text == "")
            {
                outputPath = txtAfterFilePath.Text + @"\Analysis\";
            }
            else
            {
                outputPath = txtOutputFilePath.Text + @"\Analysis\";
            }

            return outputPath;
        }

        private bool ValidInputParameters()
        {
            bool isValid = true;

            //path for before dataset
            if (txtBeforeFilePath.Text == "") {
                MessageBox.Show("Please choose directory path for the before data
points", "Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }

            //path for after dataset
            if (txtAfterFilePath.Text == "") {
                MessageBox.Show("Please choose directory path for the after data points",
"Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }

            if (txtOutputFilePath.Text == "") {
                MessageBox.Show("Please choose directory path for outputting the report
files", "Missing Directory Path", MessageBoxButtons.OK);
                isValid = false;
            }

            if (!trainingToolStripMenuItem.Checked)
            {
                //path for master suspicious key file
                if (txtMasterSuspiciousKeyFiles.Text == "")
                {
                    MessageBox.Show("Please choose directory path for the Master
Suspicious Key file", "Missing Directory Path", MessageBoxButtons.OK);
                    isValid = false;
                }

                //path for scoring file
                if (txtScoringFile.Text == "")
                {
                    MessageBox.Show("Please choose directory path for the Scoring Key
file", "Missing Directory Path", MessageBoxButtons.OK);
                    isValid = false;
                }
            }

            return isValid;
        }

        private void WriteResultsData(string path)
        {
            try
            {
                using (StreamWriter writer = new StreamWriter(path + @"\Results.txt"))
                {
                    writer.WriteLine(stopwatch.Elapsed + " seconds" +
System.Environment.NewLine +
                                "Mismatch Count: " + registryComparer.MismatchCount +
System.Environment.NewLine +
                                "Score: " + registryComparer.Score);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception writing results data file: " + e.Message);
            }
        }

        private void OpenOutputFile(string path)
```

```
{
    Process.Start(path);
}

private void updateProgress()
{
    this.progressBar1.Style = ProgressBarStyle.Marquee;
}

private void updateCompareBarMethod()
{
    this.progressBar1.Maximum = 100;
    this.progressBar1.Style = ProgressBarStyle.Blocks;
    this.progressBar1.Value = 100;
}

private void btnBrowseBeforeFile_Click(object sender, EventArgs e)
{
    string path = ConfigurationManager.AppSettings["BeforeDirectoryLastPath"];
    txtBeforeFilePath.Text = GetFolderPath(path);
}

private static void SetFilePathTextBoxProperties(TextBox textBox)
{
    textBox.ReadOnly = true;
}

private static string GetFilePath(FileDialog fileDialog)
{
    DialogResult dialogResult = fileDialog.ShowDialog();
    return fileDialog.FileName;
}

private static void SetFileFilter(FileDialog fileDialog, string fileFilter)
{
    fileDialog.Filter = fileFilter;
    fileDialog.AddExtension = true;
}

private void btnBrowseAfterFile_Click(object sender, EventArgs e)
{
    txtAfterFilePath.Text = BrowseToDirectory("AfterDirectoryLastPath");
}

private static string GetFolderPath(string path)
{
    string folderPath = "";
    FolderBrowserDialog folderBrowserDialog1 = new FolderBrowserDialog();
    folderBrowserDialog1.SelectedPath = DriveLetter.Name + path;

    if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)
    {
        folderPath = folderBrowserDialog1.SelectedPath;
    }

    return folderPath;
}
```

```
        private void btnBrowseOutputFile_Click(object sender, EventArgs e)
        {
            txtOutputFilePath.Text = BrowseToDirectory("AfterDirectoryLastPath");
        }

        private void startRegistryToolStripMenuItem_Click(object sender, EventArgs e)
        {
            StartClockAndAnalysis();
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private string BrowseToDirectory(string path)
        {
            return GetFolderPath(ConfigurationManager.AppSettings[path]);
        }

        private void browseBeforeDirectoryToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            txtBeforeFilePath.Text = BrowseToDirectory("BeforeDirectoryLastPath");
        }

        private void browseAfterDirectoryToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            txtAfterFilePath.Text = BrowseToDirectory("AfterDirectoryLastPath");
        }

        private void toolStripMenuItem1_Click(object sender, EventArgs e)
        {
            if (txtOutputFilePath.Text == "")
            {
                MessageBox.Show("Please select a parent path of where to look for the
'nonPresentKey.txt' file");
                txtOutputFilePath.Focus();
                return;
            }
            else
            {
                new ListBuilder(txtOutputFilePath.Text);

                MessageBox.Show("All Done building the suspicous dataset list");
            }

        }

        private void button2_Click(object sender, EventArgs e)
        {
            OpenFileDialog fdlg = new OpenFileDialog();

            fdlg.Title = "Browse for Master File";
            fdlg.InitialDirectory = DriveLetter.Name + @"Thesis\Before-After\Analysis\";
            fdlg.Filter = "All files (*.*)|*.*|Text files (*.txt)|*.txt";
            fdlg.FilterIndex = 2;
            fdlg.RestoreDirectory = true;

            if (fdlg.ShowDialog() == DialogResult.OK)
            {
                txtMasterSuspiciousKeyFiles.Text = fdlg.FileName;
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnBrowseScoringFile_Click(object sender, EventArgs e)
        {
            OpenFileDialog fdlg = new OpenFileDialog();

            fdlg.Title = "Browse for Scoring File";
            fdlg.InitialDirectory = DriveLetter.Name + @"Thesis\Before-After\Analysis\";
            fdlg.Filter = "All files (*.*)|*.*|Text files (*.txt)|*.txt";
            fdlg.FilterIndex = 2;
            fdlg.RestoreDirectory = true;

            if (fdlg.ShowDialog() == DialogResult.OK)
            {
                txtScoringFile.Text = fdlg.FileName;
            }
        }

        private void btnGetBeforeRegKeyData_Click(object sender, EventArgs e)
        {
            RegKeyGrabber regKeyGrabber = new RegKeyGrabber(SNAPSHOT_STATUS.BEFORE);
        }

        private void btnGetAfterRegKeyData_Click_1(object sender, EventArgs e)
        {
            RegKeyGrabber regKeyGrabber = new RegKeyGrabber(SNAPSHOT_STATUS.AFTER);
        }

        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AboutRegKeg about = new AboutRegKeg();
            about.Show();
        }

        private void getGeneralStatisticsToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (txtOutputFilePath.Text == "")
            {
                MessageBox.Show("Please select a parent path of where to look for the
'statistics.txt' file");
                txtOutputFilePath.Focus();
                return;
            }
            else
            {
                new StatisticsBuilder(txtOutputFilePath.Text);
                MessageBox.Show("All Done!!");
            }
        }

        private void mnuDetermineCode_Click(object sender, EventArgs e)
        {
            if (txtOutputFilePath.Text == "")
            {
                MessageBox.Show("Please select the directory to look for the
'statistics.txt' file");
                txtOutputFilePath.Focus();
                return;
            }
            else
            {
                StatisticsReader reader = new StatisticsReader(txtOutputFilePath.Text);
                StatisticsData statisticsData = reader.Read();
                DataAnalyzer analyzer = new DataAnalyzer(statisticsData);

                using (StreamWriter analyzerWriter = new
StreamWriter(txtOutputFilePath.Text + "\\code_classification.txt",false))
                {
                    analyzerWriter.WriteLine("code_classification: " +
analyzer.CodeClassification.ToString());
                    analyzerWriter.WriteLine("classification_percentage: " +
analyzer.ClassificationPercentage.ToString());
                }

                switch (analyzer.CodeClassification)
                   {
                    case CODE_CLASSIFICATION.CLEAN:
                        MessageBox.Show("This is a clean code!!" + Environment.NewLine +
"Percentage: " + analyzer.ClassificationPercentage );
                        break;

                    case CODE_CLASSIFICATION.MALICIOUS:
                        MessageBox.Show("This is a malicious code!!" +
Environment.NewLine + "Percentage: " + analyzer.ClassificationPercentage);
                        break;

                    case CODE_CLASSIFICATION.UNKNOWN:
                        MessageBox.Show("This is an unknown code!!" + Environment.NewLine
+ "Percentage: " + analyzer.ClassificationPercentage);
                        break;

                          default:
                        MessageBox.Show("Not sure what code is this!!" +
Environment.NewLine + "Percentage: " + analyzer.ClassificationPercentage);
                        break;
                   }
            }
        }
        private void buildNonPresentFilesToolStripMenuItem_Click(object sender, EventArgs
e)
        {

        }

        private void trainingToolStripMenuItem_Click(object sender, EventArgs e)
        {
            trainingToolStripMenuItem.Checked = true;
            experimentalToolStripMenuItem.Checked = false;
        }

        private void experimentalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            trainingToolStripMenuItem.Checked = false;
            experimentalToolStripMenuItem.Checked = true;
        }
    }
}



/*
 * File:    Form1.Designer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.groupBox1 = new System.Windows.Forms.GroupBox();
            this.btnBrowseBeforeFile = new System.Windows.Forms.Button();
            this.txtBeforeFilePath = new System.Windows.Forms.TextBox();
            this.ofdBefore = new System.Windows.Forms.OpenFileDialog();
            this.ofdAfter = new System.Windows.Forms.OpenFileDialog();
            this.groupBox2 = new System.Windows.Forms.GroupBox();
            this.btnBrowseAfterFile = new System.Windows.Forms.Button();
            this.txtAfterFilePath = new System.Windows.Forms.TextBox();
            this.groupBox3 = new System.Windows.Forms.GroupBox();
            this.btnBrowseOutputFile = new System.Windows.Forms.Button();
            this.txtOutputFilePath = new System.Windows.Forms.TextBox();
            this.sfdOutputFile = new System.Windows.Forms.SaveFileDialog();
            this.btnStartAnalysis = new System.Windows.Forms.Button();
            this.progressBar1 = new System.Windows.Forms.ProgressBar();
            this.txtStatus = new System.Windows.Forms.TextBox();
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.fileToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.browseBeforeDirectoryToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.browseAfterDirectoryToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.startRegistryToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.toolStripMenuItem1 = new System.Windows.Forms.ToolStripMenuItem();
            this.buildNonPresentFilesToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.getGeneralStatisticsToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.mnuDetermineCode = new System.Windows.Forms.ToolStripMenuItem();
            this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.helpToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.aboutToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.groupBox4 = new System.Windows.Forms.GroupBox();
            this.btnBrowseMasterFile = new System.Windows.Forms.Button();
            this.txtMasterSuspiciousKeyFiles = new System.Windows.Forms.TextBox();
            this.groupBox5 = new System.Windows.Forms.GroupBox();
            this.btnBrowseScoringFile = new System.Windows.Forms.Button();
            this.txtScoringFile = new System.Windows.Forms.TextBox();
            this.btnGetBeforeRegKeyData = new System.Windows.Forms.Button();
            this.btnGetAfterRegKeyData = new System.Windows.Forms.Button();
            this.modeToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.trainingToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.experimentalToolStripMenuItem = new
System.Windows.Forms.ToolStripMenuItem();
            this.groupBox1.SuspendLayout();
            this.groupBox2.SuspendLayout();
            this.groupBox3.SuspendLayout();
            this.menuStrip1.SuspendLayout();
            this.groupBox4.SuspendLayout();
            this.groupBox5.SuspendLayout();
            this.SuspendLayout();
            //
            // groupBox1
            //
            this.groupBox1.Controls.Add(this.btnBrowseBeforeFile);
            this.groupBox1.Controls.Add(this.txtBeforeFilePath);
            this.groupBox1.Location = new System.Drawing.Point(13, 43);
            this.groupBox1.Name = "groupBox1";
            this.groupBox1.Size = new System.Drawing.Size(494, 53);
            this.groupBox1.TabIndex = 0;
            this.groupBox1.TabStop = false;
            this.groupBox1.Text = "Path for Before Data Files";
            //
            // btnBrowseBeforeFile
            //
            this.btnBrowseBeforeFile.Location = new System.Drawing.Point(386, 18);
            this.btnBrowseBeforeFile.Name = "btnBrowseBeforeFile";
            this.btnBrowseBeforeFile.Size = new System.Drawing.Size(102, 23);
            this.btnBrowseBeforeFile.TabIndex = 1;
            this.btnBrowseBeforeFile.Text = "Browse";
            this.btnBrowseBeforeFile.UseVisualStyleBackColor = true;
            this.btnBrowseBeforeFile.Click += new
System.EventHandler(this.btnBrowseBeforeFile_Click);
            //
            // txtBeforeFilePath
            //
            this.txtBeforeFilePath.Location = new System.Drawing.Point(7, 20);
            this.txtBeforeFilePath.Name = "txtBeforeFilePath";
            this.txtBeforeFilePath.Size = new System.Drawing.Size(372, 20);
            this.txtBeforeFilePath.TabIndex = 0;
            //
            // ofdBefore
            //
            this.ofdBefore.FileName = "openFileDialog1";
            //
            // ofdAfter
            //
            this.ofdAfter.FileName = "openFileDialog1";
            //
            // groupBox2
            //
            this.groupBox2.Controls.Add(this.btnBrowseAfterFile);
            this.groupBox2.Controls.Add(this.txtAfterFilePath);
            this.groupBox2.Location = new System.Drawing.Point(12, 107);
            this.groupBox2.Name = "groupBox2";
            this.groupBox2.Size = new System.Drawing.Size(494, 53);
            this.groupBox2.TabIndex = 1;
            this.groupBox2.TabStop = false;
            this.groupBox2.Text = "Path for After Data Files";
            //
            // btnBrowseAfterFile
            //
            this.btnBrowseAfterFile.Location = new System.Drawing.Point(386, 18);
            this.btnBrowseAfterFile.Name = "btnBrowseAfterFile";
            this.btnBrowseAfterFile.Size = new System.Drawing.Size(102, 23);
            this.btnBrowseAfterFile.TabIndex = 1;
            this.btnBrowseAfterFile.Text = "Browse";
            this.btnBrowseAfterFile.UseVisualStyleBackColor = true;
            this.btnBrowseAfterFile.Click += new
System.EventHandler(this.btnBrowseAfterFile_Click);
            //
            // txtAfterFilePath
            //
            this.txtAfterFilePath.Location = new System.Drawing.Point(7, 20);
            this.txtAfterFilePath.Name = "txtAfterFilePath";
            this.txtAfterFilePath.Size = new System.Drawing.Size(372, 20);
            this.txtAfterFilePath.TabIndex = 0;
            //
            // groupBox3
            //
            this.groupBox3.Controls.Add(this.btnBrowseOutputFile);
            this.groupBox3.Controls.Add(this.txtOutputFilePath);
            this.groupBox3.Location = new System.Drawing.Point(12, 166);
            this.groupBox3.Name = "groupBox3";
            this.groupBox3.Size = new System.Drawing.Size(494, 53);
            this.groupBox3.TabIndex = 2;
            this.groupBox3.TabStop = false;
            this.groupBox3.Text = "Path for Outputting Report Files / Non Present Key
Parent Directory";
            //
            // btnBrowseOutputFile
            //
            this.btnBrowseOutputFile.Location = new System.Drawing.Point(386, 18);
            this.btnBrowseOutputFile.Name = "btnBrowseOutputFile";
            this.btnBrowseOutputFile.Size = new System.Drawing.Size(102, 23);
            this.btnBrowseOutputFile.TabIndex = 1;
            this.btnBrowseOutputFile.Text = "Browse";
            this.btnBrowseOutputFile.UseVisualStyleBackColor = true;
            this.btnBrowseOutputFile.Click += new
System.EventHandler(this.btnBrowseOutputFile_Click);
            //
            // txtOutputFilePath
            //
            this.txtOutputFilePath.Location = new System.Drawing.Point(7, 20);
            this.txtOutputFilePath.Name = "txtOutputFilePath";
            this.txtOutputFilePath.Size = new System.Drawing.Size(372, 20);
            this.txtOutputFilePath.TabIndex = 0;
            //
            // btnStartAnalysis
            //
            this.btnStartAnalysis.Location = new System.Drawing.Point(12, 410);
            this.btnStartAnalysis.Name = "btnStartAnalysis";
            this.btnStartAnalysis.Size = new System.Drawing.Size(495, 37);
            this.btnStartAnalysis.TabIndex = 3;
            this.btnStartAnalysis.Text = "Start Dataset Analysis and Reporting";
            this.btnStartAnalysis.UseVisualStyleBackColor = true;
            this.btnStartAnalysis.Click += new
System.EventHandler(this.btnStartAnalysis_Click);
            //
            // progressBar1
            //
            this.progressBar1.Location = new System.Drawing.Point(13, 351);
            this.progressBar1.Name = "progressBar1";
            this.progressBar1.Size = new System.Drawing.Size(494, 23);
            this.progressBar1.TabIndex = 4;
            //
            // txtStatus
            //
            this.txtStatus.Location = new System.Drawing.Point(13, 383);
            this.txtStatus.Name = "txtStatus";
            this.txtStatus.Size = new System.Drawing.Size(494, 20);
            this.txtStatus.TabIndex = 5;
            //
            // menuStrip1
            //
            this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.fileToolStripMenuItem,
            this.helpToolStripMenuItem});
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(517, 24);
            this.menuStrip1.TabIndex = 6;
            this.menuStrip1.Text = "menuStrip1";
            //
            // fileToolStripMenuItem
            //
            this.fileToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.browseBeforeDirectoryToolStripMenuItem,
            this.browseAfterDirectoryToolStripMenuItem,
            this.startRegistryToolStripMenuItem,
            this.toolStripMenuItem1,
            this.buildNonPresentFilesToolStripMenuItem,
            this.getGeneralStatisticsToolStripMenuItem,
            this.mnuDetermineCode,
            this.modeToolStripMenuItem,
            this.exitToolStripMenuItem});
            this.fileToolStripMenuItem.Name = "fileToolStripMenuItem";
            this.fileToolStripMenuItem.Size = new System.Drawing.Size(37, 20);
            this.fileToolStripMenuItem.Text = "File";
            //
            // browseBeforeDirectoryToolStripMenuItem
            //
            this.browseBeforeDirectoryToolStripMenuItem.Name =
"browseBeforeDirectoryToolStripMenuItem";
            this.browseBeforeDirectoryToolStripMenuItem.Size = new
System.Drawing.Size(256, 22);
            this.browseBeforeDirectoryToolStripMenuItem.Text = "Browse for the Before
Directory";
            this.browseBeforeDirectoryToolStripMenuItem.Click += new
System.EventHandler(this.browseBeforeDirectoryToolStripMenuItem_Click);
            //
            // browseAfterDirectoryToolStripMenuItem
            //
            this.browseAfterDirectoryToolStripMenuItem.Name =
"browseAfterDirectoryToolStripMenuItem";
            this.browseAfterDirectoryToolStripMenuItem.Size = new
System.Drawing.Size(256, 22);
            this.browseAfterDirectoryToolStripMenuItem.Text = "Browse for the After
Directory";
            this.browseAfterDirectoryToolStripMenuItem.Click += new
System.EventHandler(this.browseAfterDirectoryToolStripMenuItem_Click);
```

```
            //
            // startRegistryToolStripMenuItem
            //
            this.startRegistryToolStripMenuItem.Name = "startRegistryToolStripMenuItem";
            this.startRegistryToolStripMenuItem.Size = new System.Drawing.Size(256, 22);
            this.startRegistryToolStripMenuItem.Text = "Start Registry File Analysis";
            this.startRegistryToolStripMenuItem.Click += new
System.EventHandler(this.startRegistryToolStripMenuItem_Click);
            //
            // toolStripMenuItem1
            //
            this.toolStripMenuItem1.Name = "toolStripMenuItem1";
            this.toolStripMenuItem1.Size = new System.Drawing.Size(256, 22);
            this.toolStripMenuItem1.Text = "Build Suspicious Dataset List";
            this.toolStripMenuItem1.Click += new
System.EventHandler(this.toolStripMenuItem1_Click);
            //
            // buildNonPresentFilesToolStripMenuItem
            //
            this.buildNonPresentFilesToolStripMenuItem.Name =
"buildNonPresentFilesToolStripMenuItem";
            this.buildNonPresentFilesToolStripMenuItem.Size = new
System.Drawing.Size(256, 22);
            this.buildNonPresentFilesToolStripMenuItem.Text = "Build Non Present Files";
            this.buildNonPresentFilesToolStripMenuItem.Click += new
System.EventHandler(this.buildNonPresentFilesToolStripMenuItem_Click);
            //
            // getGeneralStatisticsToolStripMenuItem
            //
            this.getGeneralStatisticsToolStripMenuItem.Name =
"getGeneralStatisticsToolStripMenuItem";
            this.getGeneralStatisticsToolStripMenuItem.Size = new
System.Drawing.Size(256, 22);
            this.getGeneralStatisticsToolStripMenuItem.Text = "Build General Statistics
Data Points";
            this.getGeneralStatisticsToolStripMenuItem.Click += new
System.EventHandler(this.getGeneralStatisticsToolStripMenuItem_Click);
            //
            // mnuDetermineCode
            //
            this.mnuDetermineCode.Name = "mnuDetermineCode";
            this.mnuDetermineCode.Size = new System.Drawing.Size(256, 22);
            this.mnuDetermineCode.Text = "Classify Unknown Binary Code";
            this.mnuDetermineCode.Click += new
System.EventHandler(this.mnuDetermineCode_Click);
            //
            // exitToolStripMenuItem
            //
            this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
            this.exitToolStripMenuItem.Size = new System.Drawing.Size(256, 22);
            this.exitToolStripMenuItem.Text = "Exit";
            this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click);
            //
            // helpToolStripMenuItem
            //
            this.helpToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.aboutToolStripMenuItem});
            this.helpToolStripMenuItem.Name = "helpToolStripMenuItem";
            this.helpToolStripMenuItem.Size = new System.Drawing.Size(44, 20);
            this.helpToolStripMenuItem.Text = "Help";
            //
            // aboutToolStripMenuItem
            //
            this.aboutToolStripMenuItem.Name = "aboutToolStripMenuItem";
            this.aboutToolStripMenuItem.Size = new System.Drawing.Size(153, 22);
            this.aboutToolStripMenuItem.Text = "About Reg Keg";
            this.aboutToolStripMenuItem.Click += new
System.EventHandler(this.aboutToolStripMenuItem_Click);
            //
            // groupBox4
            //
            this.groupBox4.Controls.Add(this.btnBrowseMasterFile);
            this.groupBox4.Controls.Add(this.txtMasterSuspiciousKeyFiles);
            this.groupBox4.Location = new System.Drawing.Point(12, 225);
            this.groupBox4.Name = "groupBox4";
            this.groupBox4.Size = new System.Drawing.Size(494, 53);
            this.groupBox4.TabIndex = 2;
            this.groupBox4.TabStop = false;
            this.groupBox4.Text = "Path for Master Suspicious Key File";
            //
            // btnBrowseMasterFile
            //
            this.btnBrowseMasterFile.Location = new System.Drawing.Point(386, 18);
            this.btnBrowseMasterFile.Name = "btnBrowseMasterFile";
            this.btnBrowseMasterFile.Size = new System.Drawing.Size(102, 23);
            this.btnBrowseMasterFile.TabIndex = 1;
            this.btnBrowseMasterFile.Text = "Browse";
            this.btnBrowseMasterFile.UseVisualStyleBackColor = true;
            this.btnBrowseMasterFile.Click += new
System.EventHandler(this.button2_Click);
            //
            // txtMasterSuspiciousKeyFiles
            //
            this.txtMasterSuspiciousKeyFiles.Location = new System.Drawing.Point(7, 20);
            this.txtMasterSuspiciousKeyFiles.Name = "txtMasterSuspiciousKeyFiles";
            this.txtMasterSuspiciousKeyFiles.Size = new System.Drawing.Size(372, 20);
            this.txtMasterSuspiciousKeyFiles.TabIndex = 0;
            //
            // groupBox5
            //
            this.groupBox5.Controls.Add(this.btnBrowseScoringFile);
            this.groupBox5.Controls.Add(this.txtScoringFile);
            this.groupBox5.Location = new System.Drawing.Point(12, 284);
            this.groupBox5.Name = "groupBox5";
            this.groupBox5.Size = new System.Drawing.Size(494, 53);
            this.groupBox5.TabIndex = 7;
            this.groupBox5.TabStop = false;
            this.groupBox5.Text = "Path for Scoring Key File";
            //
            // btnBrowseScoringFile
            //
            this.btnBrowseScoringFile.Location = new System.Drawing.Point(386, 18);
            this.btnBrowseScoringFile.Name = "btnBrowseScoringFile";
            this.btnBrowseScoringFile.Size = new System.Drawing.Size(102, 23);
            this.btnBrowseScoringFile.TabIndex = 1;
            this.btnBrowseScoringFile.Text = "Browse";
            this.btnBrowseScoringFile.UseVisualStyleBackColor = true;
            this.btnBrowseScoringFile.Click += new
System.EventHandler(this.btnBrowseScoringFile_Click);
            //
            // txtScoringFile
            //
            this.txtScoringFile.Location = new System.Drawing.Point(7, 20);
            this.txtScoringFile.Name = "txtScoringFile";
            this.txtScoringFile.Size = new System.Drawing.Size(372, 20);
            this.txtScoringFile.TabIndex = 0;
            //
            // btnGetBeforeRegKeyData
            //
            this.btnGetBeforeRegKeyData.Location = new System.Drawing.Point(12, 453);
            this.btnGetBeforeRegKeyData.Name = "btnGetBeforeRegKeyData";
            this.btnGetBeforeRegKeyData.Size = new System.Drawing.Size(495, 36);
            this.btnGetBeforeRegKeyData.TabIndex = 8;
            this.btnGetBeforeRegKeyData.Text = "Create the Before Key Dataset";
            this.btnGetBeforeRegKeyData.UseVisualStyleBackColor = true;
            this.btnGetBeforeRegKeyData.Click += new
System.EventHandler(this.btnGetBeforeRegKeyData_Click);
            //
            // btnGetAfterRegKeyData
            //
            this.btnGetAfterRegKeyData.Location = new System.Drawing.Point(11, 497);
            this.btnGetAfterRegKeyData.Name = "btnGetAfterRegKeyData";
            this.btnGetAfterRegKeyData.Size = new System.Drawing.Size(495, 36);
            this.btnGetAfterRegKeyData.TabIndex = 9;
            this.btnGetAfterRegKeyData.Text = "Create the After Key Dataset";
            this.btnGetAfterRegKeyData.UseVisualStyleBackColor = true;
            this.btnGetAfterRegKeyData.Click += new
System.EventHandler(this.btnGetAfterRegKeyData_Click_1);
            //
            // modeToolStripMenuItem
            //
            this.modeToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.trainingToolStripMenuItem,
            this.experimentalToolStripMenuItem});
            this.modeToolStripMenuItem.Name = "modeToolStripMenuItem";
            this.modeToolStripMenuItem.Size = new System.Drawing.Size(256, 22);
            this.modeToolStripMenuItem.Text = "Mode";
            //
            // trainingToolStripMenuItem
            //
            this.trainingToolStripMenuItem.Name = "trainingToolStripMenuItem";
            this.trainingToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
            this.trainingToolStripMenuItem.Text = "Training";
            this.trainingToolStripMenuItem.Click += new
System.EventHandler(this.trainingToolStripMenuItem_Click);
            //
            // experimentalToolStripMenuItem
            //
            this.experimentalToolStripMenuItem.Name = "experimentalToolStripMenuItem";
            this.experimentalToolStripMenuItem.Size = new System.Drawing.Size(152, 22);
            this.experimentalToolStripMenuItem.Text = "Experimental";
            this.experimentalToolStripMenuItem.Click += new
System.EventHandler(this.experimentalToolStripMenuItem_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(517, 545);
            this.Controls.Add(this.btnGetAfterRegKeyData);
            this.Controls.Add(this.btnGetBeforeRegKeyData);
            this.Controls.Add(this.groupBox5);
            this.Controls.Add(this.groupBox4);
            this.Controls.Add(this.txtStatus);
            this.Controls.Add(this.progressBar1);
            this.Controls.Add(this.btnStartAnalysis);
            this.Controls.Add(this.groupBox3);
            this.Controls.Add(this.groupBox2);
            this.Controls.Add(this.groupBox1);
            this.Controls.Add(this.menuStrip1);
            this.MainMenuStrip = this.menuStrip1;
            this.MaximumSize = new System.Drawing.Size(525, 575);
            this.MinimumSize = new System.Drawing.Size(525, 575);
            this.Name = "Form1";
            this.Text = "Reg Keg";
            this.Load += new System.EventHandler(this.Form1_Load);
            this.groupBox1.ResumeLayout(false);
            this.groupBox1.PerformLayout();
            this.groupBox2.ResumeLayout(false);
            this.groupBox2.PerformLayout();
            this.groupBox3.ResumeLayout(false);
            this.groupBox3.PerformLayout();
            this.menuStrip1.ResumeLayout(false);
            this.menuStrip1.PerformLayout();
            this.groupBox4.ResumeLayout(false);
            this.groupBox4.PerformLayout();
            this.groupBox5.ResumeLayout(false);
            this.groupBox5.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.GroupBox groupBox1;
        private System.Windows.Forms.Button btnBrowseBeforeFile;
        private System.Windows.Forms.TextBox txtBeforeFilePath;
        private System.Windows.Forms.OpenFileDialog ofdBefore;
        private System.Windows.Forms.OpenFileDialog ofdAfter;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.Button btnBrowseAfterFile;
        private System.Windows.Forms.TextBox txtAfterFilePath;
        private System.Windows.Forms.GroupBox groupBox3;
        private System.Windows.Forms.Button btnBrowseOutputFile;
        private System.Windows.Forms.TextBox txtOutputFilePath;
        private System.Windows.Forms.SaveFileDialog sfdOutputFile;
        private System.Windows.Forms.Button btnStartAnalysis;
        private System.Windows.Forms.ProgressBar progressBar1;
        private System.Windows.Forms.TextBox txtStatus;
        private System.Windows.Forms.MenuStrip menuStrip1;
        private System.Windows.Forms.ToolStripMenuItem fileToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem
browseBeforeDirectoryToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem
browseAfterDirectoryToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem startRegistryToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem helpToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem aboutToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem toolStripMenuItem1;
        private System.Windows.Forms.GroupBox groupBox4;
        private System.Windows.Forms.Button btnBrowseMasterFile;
        private System.Windows.Forms.TextBox txtMasterSuspiciousKeyFiles;
        private System.Windows.Forms.GroupBox groupBox5;
        private System.Windows.Forms.Button btnBrowseScoringFile;
        private System.Windows.Forms.TextBox txtScoringFile;
        private System.Windows.Forms.Button btnGetBeforeRegKeyData;
        private System.Windows.Forms.ToolStripMenuItem
getGeneralStatisticsToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem mnuDetermineCode;
        private System.Windows.Forms.Button btnGetAfterRegKeyData;
        private System.Windows.Forms.ToolStripMenuItem
buildNonPresentFilesToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem modeToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem trainingToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem experimentalToolStripMenuItem;
    }
}



/*
 * File:    FrequencyScore.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public class FrequencyScore
    {
        public int Frequency { get; set; }
        public int Score { get; set; }

        public FrequencyScore(int frequecy, int score)
        {
            this.Frequency = frequecy;
            this.Score = score;
        }
    }
}



/*
 * File:    ListBuilder.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;
using RegKeg.Core;

namespace RegKeg
{
    public class ListBuilder
    {
        #region fields

        private string _parentDirectoryPath;
        private const string MASTER_FILE_NAME = "master_suspicious_keys.txt";
        private const string FREQUECY_FILE_NAME = "frequency_keys.txt";
        private const string SCORING_FILE_NAME = "scoring_keys.txt";
        private const string DISTINCT_MASTER_FILE_NAME =
"distinct_master_suspicious_keys.txt";
        private string _rootLetterDrive;

        #endregion

        #region Constructor(s)

        public ListBuilder(string parentDirectoryPath)
        {
            this._parentDirectoryPath = parentDirectoryPath + @"\";
            this._rootLetterDrive = DriveLetter.Name;

            Search();
            Dictionary<string, int> scoreSystem = ReadToMemory(this._parentDirectoryPath
+ FREQUECY_FILE_NAME);
            WriteScoringFile(scoreSystem);
        }

        private void WriteScoringFile(Dictionary<string, int> scoreSystem)
        {
            using (StreamWriter writer = new StreamWriter(this._parentDirectoryPath +
SCORING_FILE_NAME))
            {
                foreach(KeyValuePair<string, int> kvp in scoreSystem)
                {
                    writer.WriteLine(kvp.Key + "|" + kvp.Value + "|" + kvp.Value);
                }
            }
        }

        private Dictionary<string, int> ReadToMemory(string masterFile)
        {
            Dictionary<string, int> scoringSystem = new Dictionary<string, int>();
            Dictionary<string, string> cleanKeys = GetListofCleanKeys();

            string key;

            using(StreamReader reader = new StreamReader(masterFile))
            {
                while ((key = reader.ReadLine()) != null)
                {
                    if (scoringSystem.ContainsKey(key))
                    {
                        int value = scoringSystem[key] + 1;
                        scoringSystem.Remove(key);

                        scoringSystem.Add(key, value);
                    }
                    else
                    {
                        if(!cleanKeys.ContainsKey(key))
                            scoringSystem.Add(key, 1);
                    }
                }
            }

            return scoringSystem;
        }

        //compsciprof skype for professor podnar
        private Dictionary<string, string> GetListofCleanKeys()
        {
            Dictionary<string, string> cleanKeys = new Dictionary<string, string>();
            string[] lines = System.IO.File.ReadAllLines( this._rootLetterDrive +
    ConfigurationHelper.GetCleanKeyFilePath());

            foreach (string line in lines)
            {
                try
                {
                    if(!cleanKeys.ContainsKey(line))
                        cleanKeys.Add(line,"1");
                }
                catch (Exception e)
                {
                    Logger.LogError("Exception in GetListOfCleanKeys(): ", e);
                }
            }

            return cleanKeys;
        }


        public void Search()
        {
            try
            {
                string[] fileNames = Directory.GetFiles(this._parentDirectoryPath, 
"nonPresentKeys.txt", SearchOption.AllDirectories);
                List<string> uniqueKeys = new List<string>();
                List<string> duplicateKeys = new List<string>();

                using (StreamWriter masterWriter = new 
StreamWriter(this._parentDirectoryPath + MASTER_FILE_NAME, false))
                {
                    using (StreamWriter duplicateKeyWriter = new 
StreamWriter(this._parentDirectoryPath + FREQUECY_FILE_NAME, false))
                    {
                        foreach (string filename in fileNames)
                        {
                            if (filename.ToLower().EndsWith("nonpresentkeys.txt"))
                            {
                                WriteDistinctFile(filename);
                                string[] keys = System.IO.File.ReadAllLines(filename);

                                foreach (string key in keys)
                                {
                                    if (!uniqueKeys.Contains(key))
                                        uniqueKeys.Add(key);

                                    duplicateKeys.Add(key);
                                }
                            }
                        }

                        for (int i = 0; i < uniqueKeys.Count; i++)
                        {
                            masterWriter.WriteLine(uniqueKeys[i]);
                        }

                        for (int j = 0; j < duplicateKeys.Count; j++)
                        {
                            duplicateKeyWriter.WriteLine(duplicateKeys[j]);
                        }
                    }
                }
            }
            catch (Exception e)
            {
                Logger.logException("exception in Search(): " + e.Message);
            }
        }

        private void WriteDistinctFile(string fileName)
        {
            using (StreamWriter writer = new StreamWriter(this._parentDirectoryPath + 
DISTINCT_MASTER_FILE_NAME,false))
            {
                try
                {
                    writer.Write(System.IO.File.ReadAllText(fileName));
                }
                catch (Exception e)
                {
                    Logger.logException("Cannot write to distinct master file name - 
exception: " + e.Message);
                }
            }
        }

        #endregion
    }
}



/*
 * File:    Program.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Windows.Forms;

namespace RegKeg
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}



/*
 * File:    RegistryComparer.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;
using RegKeg.Core;

namespace RegKeg
{
    public class RegistryComparer
    {
        private Dictionary<string, string> _beforeDictionary;
        private Dictionary<String, String> _afterDictionary;
        private string _outputFilePath;
        private int _mismatchCount;
        private int _notPresentOnTargetDictionary;
        private Dictionary<string, int> _suspiciousKeys;
        private Dictionary<string, string> _cleanKeys;
        private Dictionary<string, FrequencyScore> _frequencyScore;
        private int _score;

        public int MismatchCount
        {
            get { return _mismatchCount; }
        }

        public int Score
        {
            get { return _score; }
        }

        public RegistryComparer(
                                Dictionary<string, string> beforeDictionary,
                                Dictionary<string, string> afterDicionary,
                                string outputFilePath,
                                Dictionary<string, FrequencyScore> frequencyScore
                                )
        {
            this._suspiciousKeys = GetListOfSuspiciousKeys();
            this._cleanKeys = GetListofCleanKeys();
            this._beforeDictionary = beforeDictionary;
            this._afterDictionary = afterDicionary;
            this._outputFilePath = outputFilePath;
            this._frequencyScore = frequencyScore;
            this._mismatchCount = 0;
            this._notPresentOnTargetDictionary = 0;
        }

        private Dictionary<string, string> GetListofCleanKeys()
        {
            Dictionary<string, string> cleanKeys = new Dictionary<string, string>();

            string[] lines = System.IO.File.ReadAllLines(DriveLetter.Name +
ConfigurationHelper.GetCleanKeyFilePath());

            foreach (string line in lines)
            {
                try
                {
                    string[] values = line.Split(',');
                    switch (values.Length)
                    {
```

```
                    case 1:
                        if(!cleanKeys.ContainsKey(values[0]))
                            cleanKeys.Add(values[0], "");
                        break;
                    case 2:
                        if(!cleanKeys.ContainsKey(values[0]))
                            cleanKeys.Add(values[0], values[1]);
                        break;

                    default:
                        Logger.logException("values of clean key is nothing");
                        break;
                }
            }
            catch (Exception e)
            {
                Logger.LogError("Exception in GetListOfCleanKeys(): ", e);
            }
        }

        return cleanKeys;
    }

    private Dictionary<string,int> GetListOfSuspiciousKeys()
    {
        Dictionary<string, int> suspiciousKeys = new Dictionary<string, int>();

        string[] lines = System.IO.File.ReadAllLines(DriveLetter.Name +
ConfigurationHelper.GetMaliciousKeyFilePath());

        foreach (string line in lines)
        {
            try
            {
                string[] values = line.Split(',');

                if (!suspiciousKeys.ContainsKey(values[0]) && values.Length == 2)
                    suspiciousKeys.Add(values[0], Convert.ToInt16(values[1]));
            }
            catch (Exception e)
            {
                MessageBox.Show("An error occurred: " + e.Message);
            }
        }

        return suspiciousKeys;
    }

    public void PerformComparison()
    {
        DeletePreviousFiles();
        this.Compare(this._beforeDictionary, this._afterDictionary, Target.Before);
        this.Compare(this._afterDictionary, this._beforeDictionary, Target.After);
    }

    private void DeletePreviousFiles()
    {
            string filePath = this._outputFilePath + "Scoring_Result.txt";

            if (File.Exists(filePath))
                File.Delete(filePath);
        }

        public void Compare(Dictionary<string,string > target, Dictionary<string,string>
lookup, Target targetDict)
        {
            string exceptionFileName = "";
            string keyNotPresentFileName = "";
            bool appendToFile;
            int maxScore = 0;
            List<int> scoringResults = new List<int>();

            if (targetDict == Target.Before)
            {
                appendToFile = false;
                exceptionFileName =
ConfigurationHelper.GetKeyByName("BeforeReg_AfterReg_exception_file");
                keyNotPresentFileName =
ConfigurationHelper.GetKeyByName("BeforeReg_AfterReg_keynotpresent_file");
            }
            else
            {
                appendToFile = true;
                exceptionFileName =
ConfigurationHelper.GetKeyByName("AfterReg_BeforeReg_exception_file");
                keyNotPresentFileName =
ConfigurationHelper.GetKeyByName("AfterReg_BeforeReg_keynotpresent_file");
            }

            if (!Directory.Exists(this._outputFilePath))
                Directory.CreateDirectory(this._outputFilePath);

            try
            {

                using (StreamWriter exceptionWriter = new
StreamWriter(this._outputFilePath + exceptionFileName))
                {
                    using (StreamWriter writer = new StreamWriter(this._outputFilePath +
"Analysis.txt"))
                    {
                        using (StreamWriter notPresentWriter = new
StreamWriter(this._outputFilePath + keyNotPresentFileName))
                        {
                            using (StreamWriter nonPresentKeys = new
StreamWriter(this._outputFilePath + "NonPresentKeys.txt"))
                            {
                                using (StreamWriter result = new
StreamWriter(this._outputFilePath + "scoring_result.txt", true))
                                {
                                    using (StreamWriter statistics = new
StreamWriter(this._outputFilePath + "statistics.txt", appendToFile))
                                    {
                                        notPresentWriter.WriteLine(targetDict ==
Target.Before
                                                                               ? "Keys Not
Present on Target (After Dictionary)"
                                                                               : "Keys Not
Present on Target (Before Dictionary)");
                                        int totalKeys = 0;

                                        foreach (string key in target.Keys)
                                        {
                                            try
                                            {
                                                // the target key is contained in the
lookup dictionary
                                                if (lookup.ContainsKey(key))
                                                {
                                                    //Values are different
                                                    if (lookup[key] != target[key])
                                                    {
                                                        if (targetDict == Target.After)
                                                        {
                                                            //Write to Analysis.txt
report file
                                                            writer.WriteLine(key +
System.Environment.NewLine);
                                                            writer.WriteLine("BEFORE: " +
System.Environment.NewLine + lookup[key]);
                                                            writer.WriteLine("AFTER: " +
System.Environment.NewLine + target[key]);
                                                            writer.WriteLine("-----------
-------------------------------------------------------------------------------------
");

                                                            this._mismatchCount++;
                                                        }
                                                    }
                                                }
                                                else
                                                {
                                                    //the key is not available in the
lookup Dictionary

                                                    //this key is found in the list of
clean keys
                                                    if (_cleanKeys.ContainsKey(key))
                                                        continue;

                                                    if (this._frequencyScore != null &&
this._frequencyScore.ContainsKey(key))
                                                    {
                                                        //Retrieve frequency from the
Dictionary for the given key
                                                        FrequencyScore scoring =
this._frequencyScore[key];
                                                        //Add this value to a running
total of frequency score
                                                        this._score += scoring.Score;
scoringResults.Add(scoring.Score);
                                                        result.WriteLine(key + "=" +
scoring.Score);

                                                        if (scoring.Score > maxScore)
                                                            maxScore = scoring.Score;

                                                        totalKeys++;
                                                    }
                                                    else
                                                    {
                                                        //not found in the frequency
dictionary
                                                        //log it to the
nonPresentKeys.txt report data file
                                                        notPresentWriter.WriteLine("Key:
" + key);

notPresentWriter.WriteLine("Value: " + target[key]);
                                                        notPresentWriter.WriteLine("-----
---------------------------------------------------");

                                                        //write to list of non present
keys - build dictionary of suspicious keys
                                                        nonPresentKeys.WriteLine(key);

this._notPresentOnTargetDictionary++;
                                                    }
                                                }
                                            }
                                            catch (Exception e)
                                            {
                                                exceptionWriter.WriteLine("Exception: " +
e);
                                            }
                                        }

                                        if (targetDict == Target.After)
                                        {
                                            if (maxScore > 0)
                                            {
                                                result.WriteLine("MaxScore=" + maxScore);
                                                statistics.WriteLine("MaxScore=" +
maxScore);
                                            }

                                            if (this._score > 0)
                                            {
                                                result.WriteLine("TotalScore=" +
this._score);
                                                statistics.WriteLine("TotalScore=" +
this._score);

                                                result.WriteLine("TotalNumberOfKeys=" +
totalKeys);
```

```
                                                statistics.WriteLine("TotalNumberOfKeys="
+ totalKeys);

                                                double average =
GetWeightByKey(this._score, this._frequencyScore);
                                                result.WriteLine("Average=" + average);
                                                statistics.WriteLine("Average=" +
average);

                                                double variance =
GetVariance(scoringResults);
                                                result.WriteLine("Variance=" + variance);
                                                statistics.WriteLine("Variance=" +
variance);

                                                double standardDeviation =
GetStandardDeviation(variance);
                                                result.WriteLine("StandardDeviation=" +
standardDeviation);
                                                statistics.WriteLine("StandardDeviation="
+ standardDeviation);
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Exception: " + e.Message);
            }
        }

        private double GetWeightByKey(int totalFrequencyScore, Dictionary<string,
FrequencyScore> frequencyScoreDictionary )
        {
            //divide the running total by the number of keys in the frequency dictionary.
            double weightByKey = totalFrequencyScore /
(double)frequencyScoreDictionary.Count;

            return weightByKey;
        }

        private double GetVariance(List<int> scoringResults)
        {
            int total = 0;
            int mean = 0;
            double squareResult = 0;
            double squaredTotal = 0;

            foreach (int s in scoringResults)
                total += s;

            try
            {
                mean = total / scoringResults.Count;

                foreach (int x in scoringResults)
                    squareResult += Math.Pow(x - mean, 2);

                squaredTotal = squareResult / scoringResults.Count;
                return (1 / (double)scoringResults.Count) * squareResult;
            }
            catch (DivideByZeroException dbze)
            {
                Logger.LogError("Divide by Zero Exception (total='" + total + "): ",
dbze);
                return 0;
            }
        }

        private double GetStandardDeviation(double variance)
        {
            return Math.Sqrt(variance);
        }
    }
}



/*
 * File:    RegistryReader.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;

namespace RegKeg
{
    public class RegistryReader
    {
        #region private fields

        private readonly string _filePath;
        private Dictionary<String, String> _registryDictionary = new Dictionary<string,
string>(120000);

        #endregion

        #region Properties

        public Dictionary<string, string> RegistryDictionary
        {
            get { return _registryDictionary; }
            set { _registryDictionary = value; }
        }
```

```
#endregion

#region Constructor(s)

public RegistryReader(string filePath)
{
    this._filePath = filePath;
    ReadFile();
}

#endregion

#region Private Methods


private void ReadFile()
{
    int lineCounter = 0;

    string key = null;
    string value = null;

    try
    {
        using (StreamReader streamReader = new StreamReader(this._filePath))
        {
            string line = "";

            while ((line = streamReader.ReadLine()) != null)
            {
                if (line == "" || lineCounter == 0)
                {
                    lineCounter++;
                    continue;
                }

                if (line.StartsWith("[") && line.EndsWith("]"))
                {
                    try
                    {
                        if(key != null)
                        {
                            this._registryDictionary.Add(key, value);
                            value = "";
                        }
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Exception: " + e.Message);
                    }
                    key = line;
                }
                else
                {
                    value += line + "\n";
                }
```

```
                        lineCounter++;
                    }
                }
            }
            catch (Exception exception)
            {
                Console.WriteLine("Exception: " + exception.Message);
            }
        }

        #endregion


    }
}



/*
 * File:    RegKeyGrabber.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Diagnostics;
using System.IO;
using System.Windows.Forms;

namespace RegKeg
{
    public class RegKeyGrabber : Form
    {
        private const string BEFORE_BATCH_FILE_PATH = @"Thesis\Before-After\before-
bruno.bat";
        private const string AFTER_BATCH_FILE_PATH = @"Thesis\Before-After\after-
bruno.bat";

        private const string BEFORE_DESTINATION_DIRECTORY = @"Thesis\Before-
After\Before\";
        private const string AFTER_DESTINATION_DIRECTORY = @"Thesis\Before-
After\Analysis\";

        private const string BEFORE_DATA_FILES_DIRECTORY = @"Thesis\Before-
After\Before\";
        private const string AFTER_DATA_FILES_DIRECTORY = @"Thesis\Before-After\After\";

        private static SNAPSHOT_STATUS _snapshotStatus;

        public RegKeyGrabber(SNAPSHOT_STATUS snapshotStatus)
        {
            _snapshotStatus = snapshotStatus;
            CallBatchFile();
            BuildSubDirectoryPath();
            ShowUserMessage();
        }

        private void ShowUserMessage()
        {
            string name =
                _snapshotStatus == SNAPSHOT_STATUS.BEFORE
                ? "before"
                : "after";

            MessageBox.Show("All done grabbing " + name + " registry keys dataset
files");
        }

        private void BuildSubDirectoryPath()
        {
            //start with count
            int count = 1;
            // get subdirectory to copy files
            string subDirectoryName = GetSubdirectoryPath(count);

            //while subdirectory exist
            while (Directory.Exists(subDirectoryName))
            {
                //increment counter
                count++;
                //try again and see if exists
                subDirectoryName = GetSubdirectoryPath(count);
            }

            //when it does not exist, create the subdirectory
            Directory.CreateDirectory(subDirectoryName);
            //Copy files to the specified subdirectory
            CopyFiles(subDirectoryName);
        }

        //Copy files to the subdirectory specified
        private static void CopyFiles(string subDirectoryName)
        {
            string path =
                _snapshotStatus == SNAPSHOT_STATUS.BEFORE
                ? BEFORE_DATA_FILES_DIRECTORY
                : AFTER_DATA_FILES_DIRECTORY;

            //Get list of all files that need to be copied
            string[] files = Directory.GetFiles(DriveLetter.Name + path);

            foreach (string f in files)
            {
                try
                {
                    //copy the file to the subdirectory specified
                    File.Copy(f, subDirectoryName + @"\" + Path.GetFileName(f));
                }
                catch (IOException e)
                {
                    Console.WriteLine("IoException: " + e.Message);
                }
            }
        }

        // Will build subdirectory string based on the naming convention of:
        //MM_DD_YYYY_XXX where xxx is a counter starting with 001.
        private static string GetSubdirectoryPath(int count)
        {
            string month = LessThan10(DateTime.Now.Month) ? "0" + DateTime.Now.Month :
DateTime.Now.Month.ToString();
            string day = LessThan10(DateTime.Now.Day) ? "0" + DateTime.Now.Day :
DateTime.Now.Day.ToString();
            string year = DateTime.Now.Year.ToString();
            string strCount = BuildCountString(count);

            string path =
                _snapshotStatus == SNAPSHOT_STATUS.BEFORE
                ? BEFORE_DESTINATION_DIRECTORY
                : AFTER_DESTINATION_DIRECTORY;

            string subDirectoryName = DriveLetter.Name + path +
                month + "-" + day + "-" + year + "_" + strCount;

            return subDirectoryName;
        }

        private static string BuildCountString(int count)
        {
            return count < 10 ? "00" + count : "0" + count;
        }

        private static bool LessThan10(int i)
        {
            return i < 10 ? true : false;
        }

        private static void CallBatchFile()
        {
            string path =
                _snapshotStatus == SNAPSHOT_STATUS.BEFORE
                ? BEFORE_BATCH_FILE_PATH
                : AFTER_BATCH_FILE_PATH;

            //create the Batch process
            Process batchProcess = new Process();
            //Set batch file name
            batchProcess.StartInfo.FileName = DriveLetter.Name + path;

            try
            {
                //start the batch process.
                batchProcess.Start();
                //wait until is done.
                batchProcess.WaitForExit();
            }
            catch (Exception e)
            {
                MessageBox.Show("Exception: " + e.Message);
            }
        }
    }
}



/*
 * File:    ScoringFileReader.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;

namespace RegKeg
{
    public class ScoringFileReader
    {
        public string FilePath { get; set; }

        public ScoringFileReader(string filePath)
        {
            this.FilePath = filePath;
        }

        public Dictionary<string, FrequencyScore> GetScoringDictionary()
        {
            Dictionary<string, FrequencyScore> scoringSystem = new Dictionary<string,
FrequencyScore>();

            try
            {
                using (StreamReader streamReader = new StreamReader(this.FilePath))
                {
                    string line = "";
                    string[] values = null;

                    while ((line = streamReader.ReadLine()) != null)
                    {
                        try
                        {
                            values = line.Split('|');
                            scoringSystem.Add(values[0], new FrequencyScore(

Convert.ToInt32(values[1]),

Convert.ToInt32(values[2])
                                                                             ));
                        }
                        catch (Exception e)
                        {
                            Console.WriteLine("Exception: " + e.Message);
                        }
                    }
                }
            }
            catch (Exception exception)
            {
                Console.WriteLine("Exception: " + exception.Message);
            }

            return scoringSystem;
        }
    }
}



/*
 * File:    SNAPSHOT_STATUS.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public enum SNAPSHOT_STATUS
    {
        BEFORE,
        AFTER,
        UNKNOWN
    }
}



/*
 * File:    StatisticsBuilder.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using RegKeg.Core;

namespace RegKeg
{
    public class StatisticsBuilder
    {
        #region Private Fields

        private string _parentDirectoryPath;
        private string _rootLetterDrive;

        private const string GRAND_TOTAL_STATISTICS = "grand_total_statistics.txt";
        private const string CLEAN_CODE_STATISTICS = "clean_code_total_statistics.txt";
        private const string MALICIOUS_CODE_STATISTICS = 
"malicious_code_total_statistics.txt";

        private int _totalScore;
        private int _maxScore;
        private double _average;
        private double _variance;
        private double _standardDeviation;

        #endregion

        #region Constructor(s)

        public StatisticsBuilder(string parentDirectoryPath)
        {
            this._parentDirectoryPath = parentDirectoryPath + @"\";
            this._rootLetterDrive = DriveLetter.Name;

            this.Search(STATS_MODE.CLEAN);
            this.Search(STATS_MODE.MALICIOUS);
            this.Search(STATS_MODE.ALL);
        }

        #endregion

        #region Private Method(s)

        private void Search(STATS_MODE mode)
        {
            try
            {
                if (mode == STATS_MODE.ALL)
                {
                    List<string> dataFiles = GetAllDataFiles();
                    GetDataFromFiles(dataFiles);
                    WriteGrandTotal(GRAND_TOTAL_STATISTICS);
                    WriteCsvStatisticsData(GetStatisticsData(), this._parentDirectoryPath 
+ @"statistics.csv");
                }

                if(mode == STATS_MODE.CLEAN)
                {
                    if (Directory.Exists(this._parentDirectoryPath)) {

                        List<string> cleanDataFiles = GetCleanDataFiles();
                        GetDataFromFiles(cleanDataFiles);
                        WriteGrandTotal(CLEAN_CODE_STATISTICS);
                    }
                }
                if (mode == STATS_MODE.MALICIOUS)
                {
                    List<string> maliciousFile = GetMaliciousDataFiles();
                    GetDataFromFiles(maliciousFile);
                    WriteGrandTotal(MALICIOUS_CODE_STATISTICS);
                }

                ClearTotals();
            }
            catch (Exception e)
            {
                Logger.logException("exception in StatisticsBuilder.Search(): " +
e.Message);
            }
        }

        private List<string> GetMaliciousDataFiles()
        {
            List<string> cleanDataFiles = GetCleanDataFiles();
            List<string> allDataFiles = GetAllDataFiles();

            List<string> maliciousFile = new List<string>();

            foreach (string path in allDataFiles)
            {
                if (!cleanDataFiles.Contains(path))
                {
                    maliciousFile.Add(path);
                }
            }

            return maliciousFile;
        }

        private List<string> GetAllDataFiles()
        {
            List<string> dataFiles = new List<string>();

            string[] fileNames = Directory.GetFiles(this._parentDirectoryPath,
"statistics.txt", SearchOption.AllDirectories);
            dataFiles = AddToDataFiles(dataFiles, fileNames);

            return dataFiles;
        }

        private List<string> GetCleanDataFiles()
        {
            List<string> dataFiles = new List<string>();
            DirectoryInfo directoryInfo = new DirectoryInfo(this._parentDirectoryPath);

            IEnumerable<DirectoryInfo> matchingDirs = directoryInfo
                .EnumerateDirectories("*.*", System.IO.SearchOption.AllDirectories)
                .Where(d => d.EnumerateFiles("Clean_Code.txt").Any());

            foreach (DirectoryInfo info in matchingDirs)
            {
                FileInfo[] fileInfo = info.GetFiles("statistics.txt",
SearchOption.AllDirectories);

                foreach (FileInfo f in fileInfo)
                {
                    string[] fileNames = Directory.GetFiles(f.Directory.ToString(),
"statistics.txt", SearchOption.AllDirectories);
                    dataFiles = AddToDataFiles(dataFiles, fileNames);
                }
            }
            return dataFiles;
        }

        private void WriteCsvStatisticsData<T>(IEnumerable<T> items, string path)
        {
            Type itemType = typeof(T);
            var props = itemType.GetProperties(BindingFlags.Public |
BindingFlags.Instance)
                                .OrderBy(p => p.Name);

            using (var writer = new StreamWriter(path))
            {
                writer.WriteLine(string.Join(", ", props.Select(p => p.Name)));

                foreach (var item in items)
                {
                    writer.WriteLine(string.Join(", ", props.Select(p => p.GetValue(item,
null))));
                }
            }
        }

        private void ClearTotals()
        {
            this._totalScore = 0;
            this._maxScore = 0;
            this._average = 0;
            this._variance = 0;
            this._standardDeviation = 0;
        }

        private List<string> AddToDataFiles(List<string> dataFiles, string[] fileNames)
        {
            foreach (string f in fileNames)
            {
                dataFiles.Add(f);
            }

            return dataFiles;
        }

        private List<StatisticsData> GetStatisticsData()
        {
            List<string> allDataFiles = GetAllDataFiles();
            List<string> cleanDataFiles = GetCleanDataFiles();
            List<string> maliciousDataFiles = GetMaliciousDataFiles();
            List<StatisticsData> statisticsDataFiles = new List<StatisticsData>();

            foreach (string filename in allDataFiles)
            {
                if (filename.EndsWith("statistics.txt"))
                {
                    StatisticsData statisticsData = new StatisticsData();

                    statisticsData.FileName = filename;
                    statisticsData.IsMaliciousCode =
maliciousDataFiles.Contains(filename);

                    string[] lines = System.IO.File.ReadAllLines(filename);

                    foreach (string line in lines)
                    {
                        string[] keyValuePair = line.Split('=');

                        try
                        {
                            switch (keyValuePair[0])
                            {
                                case "TotalScore":
                                    statisticsData.TotalScore =
Convert.ToInt16(keyValuePair[1]);
                                    break;

                                case "MaxScore":
                                    statisticsData.MaxScore =
Convert.ToInt32(keyValuePair[1]);
                                    break;

                                case "Average":
                                    statisticsData.Average =
Convert.ToDouble(keyValuePair[1]);
                                    break;

                                case "Variance":
                                    statisticsData.Variance =
Convert.ToDouble(keyValuePair[1]);
                                    break;

                                case "StandardDeviation":
                                    statisticsData.StandardDeviation =
Convert.ToDouble(keyValuePair[1]);
                                    break;

                                default:
                                    Logger.LogError("Statistics.Search(): ", new
Exception("Can't find statistics variable in switch statement"));
                                    break;
                            }
                        }
                        catch (Exception e)
                        {
                            Logger.LogError("error in Search() - try conversion: ", e);
                        }
                    }
                    statisticsDataFiles.Add(statisticsData);
                }
            }
            return statisticsDataFiles;
        }

        private void GetDataFromFiles(List<string> fileNames)
        {
            foreach (string filename in fileNames)
            {
                if (filename.EndsWith("statistics.txt"))
                {
                    string[] lines = System.IO.File.ReadAllLines(filename);

                    foreach (string line in lines)
                    {
                        string[] keyValuePair = line.Split('=');

                        try
                        {
                            switch (keyValuePair[0])
                            {
                                case "TotalScore":
                                    this._totalScore += Convert.ToInt16(keyValuePair[1]);
                                    break;

                                case "MaxScore":
                                    this._maxScore += Convert.ToInt32(keyValuePair[1]);
                                    break;

                                case "Average":
                                    this._average += Convert.ToDouble(keyValuePair[1]);
                                    break;

                                case "Variance":
                                    this._variance += Convert.ToDouble(keyValuePair[1]);
                                    break;

                                case "StandardDeviation":
                                    this._standardDeviation +=
    Convert.ToDouble(keyValuePair[1]);
                                    break;

                                default:
                                    Logger.LogError("Statistics.Search(): ", new
    Exception("Can't find statistics variable in switch statement"));
                                    break;
                            }
                        }
                        catch (Exception e)
                        {
                            Logger.LogError("error in Search() - try conversion: ", e);
                        }
                    }
                }
            }
        }

        private void WriteGrandTotal(string fileName)
        {
            using (StreamWriter writer = new StreamWriter(this._parentDirectoryPath +
fileName,false))
            {
                writer.WriteLine("TotalScore=" + _totalScore);
                writer.WriteLine("MaxScore=" + _maxScore);
                writer.WriteLine("Average=" + _average);
                writer.WriteLine("Variance=" + _variance);
                writer.WriteLine("StandardDeviation=" + _standardDeviation);
            }
        }

        #endregion
    }
}



/*
 * File:    StatisticsData.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    public class StatisticsData
    {
        #region Private Fields

        private string _fileName;
        private bool _isMaliciousCode;
        private int _totalScore;
        private int _maxScore;
        private double _average;
        private double _variance;
        private double _standardDeviation;

        #endregion

        #region Properties

        public string FileName
        {
            get { return this._fileName; }
            set { this._fileName = value; }
        }

        public bool IsMaliciousCode
        {
            get { return this._isMaliciousCode; }
            set { this._isMaliciousCode = value; }
        }

        public int TotalScore
        {
            get { return this._totalScore; }
            set { this._totalScore = value; }
        }

        public int MaxScore
        {
            get { return this._maxScore; }
            set { this._maxScore = value; }
        }

        public double Average
        {
            get { return this._average; }
            set { this._average = value; }
        }

        public double Variance
        {
            get { return this._variance; }
            set { this._variance = value; }
        }

        public double StandardDeviation
        {
            get { return this._standardDeviation; }
            set { this._standardDeviation = value; }
        }

        #endregion

        #region Constructor(s)

        public StatisticsData()
        {

        }

        #endregion
    }
}



/*
 * File:    StatisticsReader.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.IO;
using RegKeg.Core;

namespace RegKeg
{
    public class StatisticsReader
    {
        #region Private Field(s)

        private string _parentDirectoryPath;
        private string _rootLetterDrive;

        #endregion

        #region Properties

        public string ParentDirectoryPath
        {
            get { return this._parentDirectoryPath; }
            set { this._parentDirectoryPath = value; }
        }

        public string RootLetterDrive
        {
            get { return this._rootLetterDrive; }
            set { this._rootLetterDrive = value; }
        }

        #endregion

        #region Constructor(s)

        public StatisticsReader(string parentDirectoryPath)
        {
            this._parentDirectoryPath = parentDirectoryPath + @"\";
            this._rootLetterDrive = DriveLetter.Name;
        }

        #endregion

        public StatisticsData Read()
        {
            StatisticsData statisticsData = new StatisticsData();
            string[] fileNames = Directory.GetFiles(this._parentDirectoryPath,
"statistics.txt", SearchOption.AllDirectories);

            foreach (string filename in fileNames)
            {
                if (filename.EndsWith("statistics.txt"))
                {
                    string[] lines = System.IO.File.ReadAllLines(filename);

                    foreach (string line in lines)
                    {
                        string[] keyValuePair = line.Split('=');

                        try
                        {
                            switch (keyValuePair[0])
                            {
                                case "TotalScore":
                                    statisticsData.TotalScore =
Convert.ToInt16(keyValuePair[1]);
                                    break;

                                case "MaxScore":
                                    statisticsData.MaxScore =
Convert.ToInt32(keyValuePair[1]);
                                    break;

                                case "Average":
                                    statisticsData.Average =
Convert.ToDouble(keyValuePair[1]);
                                    break;

                                case "Variance":
                                    statisticsData.Variance =
Convert.ToDouble(keyValuePair[1]);
                                    break;

                                case "StandardDeviation":
                                    statisticsData.StandardDeviation =
Convert.ToDouble(keyValuePair[1]);
                                    break;

                                default:
                                    Logger.LogError("Statistics.Search(): ", new
Exception("Can't find statistics variable in switch statement"));
                                    break;
                            }
                        }
                        catch (Exception e)
                        {
                            Logger.LogError("error in Search() - try conversion: ", e);
                        }
                    }
                }
            }
            return statisticsData;
        }
    }
}



/*
 * File:    STATS_MODE.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    enum STATS_MODE
    {
        ALL,
        CLEAN,
        MALICIOUS
    }
}



/*
 * File:    STATS_MODE.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

namespace RegKeg
{
    enum STATS_MODE
    {
        ALL,
        CLEAN,
        MALICIOUS
    }
}



/*
 * File:    AssemblyInfo.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System.Reflection;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("RegKeg.Core")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("RegKeg.Core")]
[assembly: AssemblyCopyright("Copyright ©  2012")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]
```

```
// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("0ed914c9-ce1f-4ba7-946b-78d81635fce3")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]



/*
 * File:    ConfigurationHelper.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;

namespace RegKeg.Core
{
    public static class ConfigurationHelper
    {
        public static string GetKeyByName(string keyName)
        {
            try
            {
                return
System.Configuration.ConfigurationManager.AppSettings[keyName].ToString();
            }
            catch (Exception e)
            {
                Logger.LogError(e);
                return "";
            }
        }

        public static string GetMaliciousKeyFilePath()
        {
            return GetKeyByName("MalicousKeyFilePath");
        }

        public static string GetCleanKeyFilePath()
        {
            return GetKeyByName("CleanKeyFilePath");
        }
    }
}
```

```
/*
 * File:    Logger.cs
 * Project: Master's Thesis
 * Author:  Bruno Nader
 * Adviser: Dr. Hrvoje Podnar
 * Date:    May 20, 2014
 */

using System;
using System.IO;

namespace RegKeg.Core
{
    public static class Logger
    {
        #region Private Constants

        private const string FILE_NAME = "Exceptions.log";

        #endregion

        #region Public Static Methods

        public static void LogError(Exception e)
        {
            LogError("", e);
        }

        public static void LogError(string message, Exception e)
        {
            using (StreamWriter exceptionWriter = new StreamWriter(FILE_NAME))
            {
                try
                {
                    exceptionWriter.WriteLine(DateTime.Now + " - " + message);
                    exceptionWriter.WriteLine("Exception Message: " + e.Message);
                }
                catch(Exception){ }
            }
        }

        public static void logException(string message)
        {
            using (StreamWriter exceptionWriter = new StreamWriter(FILE_NAME))
            {
                try
                {
                    exceptionWriter.WriteLine(DateTime.Now + " - " + message);
                }
                catch (Exception) { }
            }
        }
        #endregion
    }
}
```

REFERENCES

[1] D. Sanok Jr, "An Analysis of How Antivirus Methodologies Are Utilized in Protecting Computers from Malicious Code", in Proceedings of the 2nd annual conference on Information Security curriculum development, Kennesaw, GA, USA, 2005, pp. 142-144

[2] M. Preda, M. Christodorescu, S. Jha, S. Debray, "A Semantics-Based Approach to Malware Detection". *ACM Transactions on Programming Languages and Systems*, ACM, vol. 30, No. 5, Article 25, August 2008

[3] Statistics Brain, "Computer Virus Statistics", [Online]. Available: http://www.statisticbrain.com/computer-virus-statistics/. [Retrieved February 2013].

[4] Wikipedia, "Computer Virus", [Online]. Available: http://en.wikipedia.org/wiki/Computer_virus. [Retrieved February 2013]

[5] J. Bergeron, M. Debbabi, M. Erhioui, B. Ktari, "Static Analysis of Binary Code to Isolate Malicious Behaviors" in IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Stanford, CA, USA, 1999, pp. 184-189

[6] A. Lee, V. Varadharajan, U. Tupakula, "On Malware Characterization and Attack Classification" in Proceedings of the First Australian Web Conference, Adelaide, Australia, 2013

[7] Symantec Corp., “Trojan Horse – Symantec Security Response - Glossary”, [Online]. Available: http://us.norton.com/security_response/glossary/define.jsp?letter=t&word=trojan-horse. [Retrieved February 2013]

[8] D. Distler, “Malware Analysis: An Introduction”. [Online]. Available: http://www.sans.org/reading-room/whitepapers/malicious/malware-analysis-introduction-2103. [Retrieved February 2013]

[9] Wikipedia, “Adware”, [Online]. Available: http://en.wikipedia.org/wiki/Adware. [Retrieved February 2013]

[10] J. Crandall, R. Ensafi, S. Forrest, J. Ladau, B. Shebaro, “The Ecology of Malware” in NSPW ’08 Proceedings of the 2008 workshop on New security paradigms, pp. 99-106.

[11] Wikipedia, “Computer Worm”, [Online]. Available: http://en.wikipedia.org/wiki/Computer_worm. [Retrieved February 2013]

[12] J. Rabek, R. Khazan, S. Lewandowski, R. Cunningham. Detection of Injected, Dynamically Generated, and Obfuscated Malicious Code. *Proceedings of the 2003 ACM workshop on Rapid malcode. ACM*, pp-76-82. October 2003

[13] C. Willems, T. Holz, F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *Security & Privacy, IEEE,* vol. 5, pp 32-39. April 2007

[14] L. Nataraj, P. Porras, V. Yegneswaran, J. Zhang. A Comparative Assessment of Malware Classification using Binary Texture Analysis and Dynamic Analysis. [Online]. Available: http://vision.ece.ucsb.edu/publications/aisec17-nataraj.pdf

[15] U. Bayer, A. Moser, C. Kruegel, E. Kirda. (2006, August). Dynamic Analysis of Malicious Code. *Journal in Computer Virology* [Online]. *2(1)*, pp. 67-77. Available: http://www.springerlink.com/content/7023515231t17557/fulltext.pdf

[16] CWSandbox, “Behavior-based Malware Analysis,” [Online]. Available: http://www.cwsandbox.org/. [Retrieved December 2009].

[17] Norman, Norman Sandbox, [Online]. Available: http://www.norman.com/. [Retrieved December 2009].

[18] TTanalyze, “A tool for analyzing malware,” [Online]. Available: http://www.seclab.tuwien.ac.at/projects/ttanalyze/. [Retrieved December 2009]

[19] A. Vasudevan, R. Yerraballi, “Cobra: Fine-grained malware analysis using stealth localized-execution,” *Security and Privacy,* vol. 21, no. 24, pp. 15-279, May, 2006.

[20] A. Moser, C. Kruegel, E. Kirda, “Exploring Multiple Execution Paths for Malware Analysis”, in *Proceedings of the 2007 IEEE Symposium on Security and Privacy,* 2007, pp. 231-245

[21] A. Moser, C. Kruegel, E. Kirda, “Limits of Static Analysis for Malware Detection”, *23rd Annual Computer Security Applications Conference.*

[22] Wikipedia, “Microsoft Visual Studio”, [Online]. Available: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio. [Retrieved March 2014]

[23] Wikipedia, “C Sharp (Programming Language)”, [Online]. Available: http://en.wikipedia.org/wiki/C_Sharp_(programming_language). [Retrieved March 2014]

[24] GParted, “GParted – Documentation”, [Online]. Available: http://gparted.org/documentation.php. [Retrieved March 2014]

[25] Clonezilla, “The Free and Open Source Software for Disk Imaging and Cloning”, [Online]. Available: http://clonezilla.org. [Retrieved March 2014]

[26] Wikipedia, “Windows Registry”, [Online]. Available: http://en.wikipedia.org/wiki/Windows_Registry/. [Retrieved March 2013]

[27] Problem Solving with Algorithms and Data Structures. [Online]. Available: http://www.cs.umd.edu/class/spring2004/cmsc420/sp04-part2v2/node4.html . [Retrieved April 2014]

[28] Problem Solving with Algorithms and Data Structures. [Online]. Available: http://www.cs.umd.edu/class/spring2004/cmsc420/sp04-part2v2/node4.html . [Retrieved April 2014]

[29] Wikipedia, "pushd and popd", [Online]. Available: http://en.wikipedia.org/wiki/Pushd_and_popd. [Retrieved November 2013]

[30] Computer Hope, "Microsoft DOS dir command", [Online]. Available: http://www.computerhope.com/dirhlp.htm. [Retrieved November 2013]